TOTAL VARIATION DIMINISHING WEIGHTED ESSENTIALLY NON-OSCILLATORY (WENO) THIRD ORDER FINITE DIFFERENCE SCHEMES FOR APPROXIMATING STEADY STATE SOLUTIONS TO SCALAR HYPERBOLIC CONSERVATION LAWS: A CONVERGENCE STUDY

Author

Charles Michael Anthony Baird

Department of Mathematical Sciences

Indiana University of South Bend

Advisor

Shanqin Chen, Ph.D.

Department of Mathematical Sciences

Committee

Dana Vrajitoru, Ph.D.

Department of Computer Science and Informatics

Yu Song, Ph.D.

Department of Mathematical Sciences

Director

Liqiang Zhang, Ph.D.

Department of Computer Science and Informatics

Submitted to the faculty of Indiana University South Bend in partial fulfillment of the requirements for the degree of

Master of Science in Applied Mathematics and Computer Science

Spring 2019

Accepted by the Graduate Faculty of Indiana University South Bend's Department of Mathematical Sciences and the Department of Computer Science and Informatics, in partial fulfillment for the degree of Master of Science in Applied Mathematics and Computer Science

_____ _____

Advisor: Shanqin Chen, Ph.D. Date

_____ _____

Committee Member: Yu Song, Ph.D. Date

_____ _____

Committee Member: Dana Vrajitoru, Ph.D. Date

_____ _____

Director: Liqiang Zhang, Ph.D. Date

ABSTRACT

In the study of Computational Fluid Dynamics there is a fundamental family of partial differential equations known as hyperbolic conservation laws which are used to describe transport phenomena such as advection and propagation. Solutions to these equations can sometimes incorporate functional discontinuities which prove intractable for standard finite difference schemes to resolve, if they even converge to a solution at all, which normally occurs at the cost of setting an infeasibly small time step discretization and is thus computationally unfavorable. Weighted Essentially Non-Oscillatory (WENO) finite difference schemes were thus developed in [8] and subsequently improved upon in [6, 7, 10, 18] and others in order to provide high order accurate solutions to such equations involving strong shocks and/or discontinuities. Total Variation Diminishing time-marching and Successive Over-Relaxation Gauss-Seidel schemes can be utilized within the WENO framework to provide accelerated convergence potential [5]. Through the use of Mathworks' Matlab programming suite various examples of 1- and 2-dimensional nonlinear initial condition and boundary value problems are shown to be solved using a third-order accurate time-marching and fast-sweeping WENO algorithm. A comparison of these time-marching and fast-sweeping methods is then made to exemplify the degree to which convergence can be accelerated while still maintaining uniform high order accuracy.

ACKNOWLEDGEMENTS

Thank you Shanqin Chen, my advisor and professor, for the seminal role you have played in making this entire enterprise possible. Thank you Dana Vrajitoru and Yu Song for assembling the basis of my dissertation committee. Thank you Monika Lynker for your counsel at rather pivotal moments in my undergraduate and graduate careers. Thank you to the myriad faculty of Indiana University South Bend's Departments of Physics and Mathematical Sciences collectively for providing quite a special place for me to achieve my Undergraduate and subsequent Master degrees. Thank you Chi Wang Shu for your prolific works and brimming enthusiasm for the subject matter of this paper. Thank you to my father and mother, Michael L. Baird and Anne E. Born, for your endearing and unparalleled support. Thank you to my friend Milo F. DiPaola, for your intellectual prowess and unrelenting dynamism in all things mathematics, both of which have been an honor and a pleasure to even be around. And an amorphous but yet incredibly indebted thank you to all of the great mathematical minds that have so dotted human history, whose efforts have been the stalwart wave front that has allowed my vessel to reach such distant shores as these.

TABLE OF CONTENTS

FOREWORD

Throughout the production of this work, the ultimate aim had always been to understand, implement, and analyze two separate finite difference schemes for solving the steady state of advection dominated partial differential equations; to compare their respective rates of convergence and to show increased efficiency. Over time this original goal bifurcated enough to include a discussion on CFL conditions and the possibility for post-processing analysis to enhance resolution in the areas of estimated discontinuities. As these later developments are tangential to the convergence study, only Examples 1 and 6 receive any discussions of CFL conditions, and only Examples 2 and 3 are given any post-processing resolution enhancement near the shock location. Examples 4 and 5 simply display the estimated discontinuous regions, as a segment and a point. A more rigorous investigation with such post-processing foremost in mind would be quite interesting, but is beyond the scope of this thesis.

The figures and tables of this thesis will thus tell two different stories. The figures for each example showcase ideas surrounding either CFL condition manipulation or discontinuity estimation. The tables are then concerned with the iteration rates of convergence, along with calculating the order of convergence across various grid discretizations, to verify third-order accuracy. The final results are a comparison of worst and best iteration counts to achieve convergence.

Altogether this is meant to be a self-contained and comprehensive dissertation on the ideas of finite difference methods, the concepts of weighted essentially non-oscillatory schemes, Runge-Kutta third order time-marching and Gauss-Seidel fast-sweeping methods. The mathematics and the computer code are extensively represented in this thesis, and to make the coding components less daunting, a pseudo-code example is given in the Appendix section.

-cb

1. INTRODUCTION

PREFACE

Within the field of numerical analysis, algorithmic schemes which yield high-order accurate approximations of weak solutions to systems of partial differential equations are crucially important assets in the study of Computational Fluid Dynamics in particular, and in the creation of physical modelling systems in general. To simulate real-world physical phenomena within the confines of a computerized architecture must surely be considered to be one of the major achievements of 20^{th} Century scientific advancement, and the awe-inspiring products of such achievements should be understood to be reflections of the incredible ingenuity of the algorithms employed to produce them. These algorithms, although of monumental utility and applicability, are imperfect by design, and, dependent upon the scenarios for which they are used, these imperfections can become manifest to such a degree so as to ruin the efficacy of the algorithm completely. A hypothetical *dream* algorithm would be one which offers precision to an arbitrary degree, accomplishing such accuracy without any exorbitant toll on resources, with the ability to solve entire swaths of similar problems without prejudice. Concerning finite difference algorithms, such a hypothetical *dream* is just that; however there are meaningful adaptations which can be made to the core schematic which help ameliorate these functional drawbacks under specific conditions.

Now, various numerical approximation algorithms may prove to be better suited in practice for solving any particular system of partial differential equations, with such contenders as finite volume, finite element, hybrid discontinuous Galerkin methods, etc. Finite difference schemes—albeit primitive algorithms by comparison—can still prove to be of considerable utility, despite their relative simplicity, and in certain cases can achieve parity in accuracy and iteration count with other more extensive frameworks [2, 4]. What is important in the successful application of such finite difference methods (i.e. that convergence is possible at a manageable exhaustion of time and resources, at the desired accuracy) is that for one, the specific partial differential equations to be solved are *friendly* to the abstractions of the finite difference framework, and that two, the solutions themselves are equally, *friendly*. Meaning that the equations themselves refer to quantities with rates of change which admit to some representation in a discretized limit definition form, and that the solutions to such equations are equally

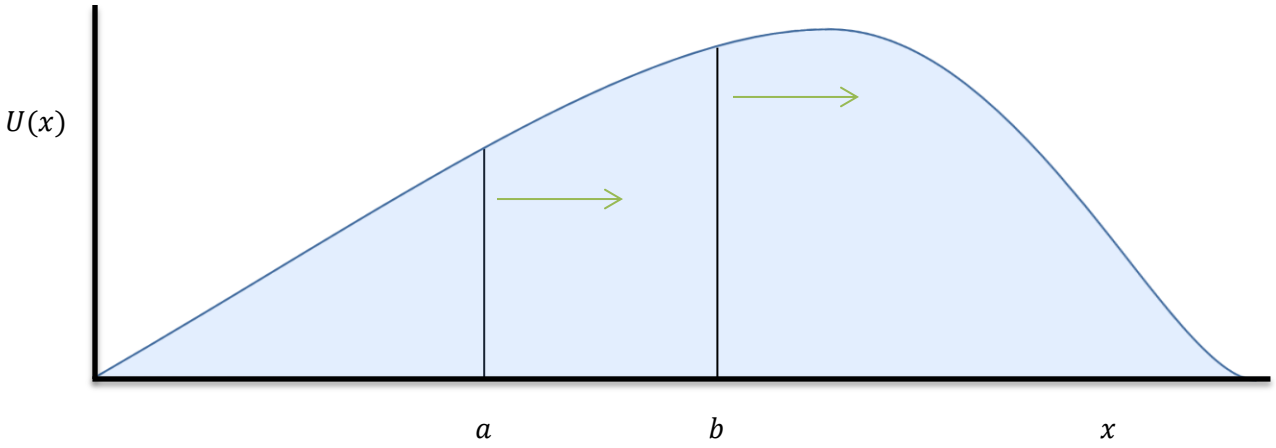
well established given this discretization. To elucidate the former a bit, the following section will provide some exposition on the matter of scalar hyperbolic conservation laws: the family of partial differential equations which this paper is primarily concerned with. To then clarify the latter, the final section of this introduction will detail the methodology of finite difference algorithms in general, including a rudimentary derivation of such schemes. It will end with some of the central limitations to the more standard formulations of finite difference schemes, in order to provide the necessary motivations behind the development of Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory schemes.

SCALAR HYPERBOLIC CONSERVATION LAWS

A fundamental component to any isomorphic abstraction of the dynamics of physically realizable systems into a set of differential equations is the requirement that such equations obey the laws of conservation. If we consider the propagation of a wave, or the flow of traffic, or the movement of gas particles within a container, or the advection of heat, then it must be foremost assumed that the *density* of these quantities at any particular location is entirely proportional to the amount to which these quantities have or will *flux* through the region bounding this location. As is often stated in physics, nothing is created or destroyed in these systems (the height of a wave cannot spontaneously increase or decrease from nowhere, for instance, but can do so only because of the dynamics of the neighboring wave heights), and because of this restriction, the equations which describe these phenomena must *conserve* the relative quantities therein [1, 2, 3, 4, 20]. The canonical formalization of such a conservation law written in advection form is presented below.

$$U_t + f(U)_x = \phi \tag{1.1}$$

The function f in (1.1) defines the fluxing of the quantity U, where ϕ denotes the amount of source or sink present. If we assume the absence of a source term, we can perform some rough integration of (1.1) in order to stress the notion of the fluxing of U in deriving a solution, over an arbitrary interval $[a, b]$.



[FIGURE 1]

An arbitrary 1-dimensional function $U(x)$ with dynamics that can be understood in an advection dominated framework. Assuming $U(x)$ models the propagation of a right-moving wave, the density of $U(x)$ within the interval $[a, b]$ can be obtained by examining the aggregate fluxing of $U(x)$ across this interval, i.e. the inflow at $x = a$ minus the outflow at $x = b$.

1.2) $$\int_a^b U(x,t)_t dx = -\int_a^b f\big(U(x,t)\big)_x dx$$

1.3) $$\int_a^b U(x,t)_t dx = f\big(U(a,t)\big) - f\big(U(b,t)\big) = (inflow\ at\ a) - (outflow\ at\ b)$$

As a means to illuminate the field of relevant players in this story, listed below are some of the more familiar partial differential equations which belong to this family of conservation laws [2].

1.4) $U_t + cU_x = 0$ [Advection]

1.5) $U_{tt} - c^2 U_{xx} = 0$ [Wave]

1.6) $\rho_t + \nabla \cdot J = 0$ [Continuity]

1.7) $U_t + \frac{1}{2}U^2{}_x = 0$ [Inviscid Burgers]

By representing conserved quantities whose dynamics are entirely dictated by a fluxing of relative density across some arbitrary interval, the above equations readily translate themselves over to a discretized framework of finite difference schemes which employ some applicable fluxing model. To focus in on and demystify the terminology behind *hyperbolic* conservation laws in particular [4], of which (1.5) and (1.7) are examples, two levels of description can be offered, both of which refer to similar basic shared structures, just stated in different ways. The first pertains to the value of the discriminant of a generalized second order linear partial differential equation, given below.

1.8) $$AU_{tt} + BU_{xt} + CU_{xx} + DU_t + EU_x + FU + G = 0$$

1.9) $$\begin{cases} B^2 - 4AC < 0 & Elliptic \\ B^2 - 4AC = 0 & Parabolic \\ B^2 - 4AC > 0 & Hyperbolic \end{cases}$$

Another mode in which to understand the categorization of such equations is by examining the set of eigenvalues generated by the associated Jacobian matrix for a given system of conservation laws (whereby the assumed *system of conservation laws* thus far has been considered to be systems of single laws [where, for example, the Euler equations in convective form is a system of three conservation laws, referring to the conservation of mass, momentum, and energy respectively]). In terms of an n x n system of conservation laws

1.10) $$\begin{cases} U_t^1 + f^1(U^1, \dots, U^n)_x = 0 \\ \vdots \\ U_t^n + f^n(U^1, \dots, U^n)_x = 0 \end{cases}$$

we can write a condensed form of such a system as

1.11) $$U_t + A(U)U_x = 0$$

1.12) $$A(U) = \begin{pmatrix} \frac{\partial f^1}{\partial U^1} & \cdots & \frac{\partial f^1}{\partial U^n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f^n}{\partial U^1} & \cdots & \frac{\partial f^n}{\partial U^n} \end{pmatrix}$$

The given system of conservations laws is then considered to be strictly hyperbolic if the Jacobian matrix (1.12) has n real and distinct eigenvalues, ranging from

1.13) $$\lambda^1(U) < \cdots < \lambda^n(U)$$

The final term demanding insight is the *scalar* in *scalar hyperbolic conservation laws*. In (1.1) the presented equation is already in the form of a scalar conservation law, but to provide some context, consider the nonlinear transport equation

1.14) $$U_t + a(x,t)U_x = 0$$

1.15) $$a(x,t) = U(x,t)$$

Then (1.14) can be rewritten as

1.16) $$U_t + UU_x = 0$$

which can be rewritten further still in conservative form as

1.17) $$U_t + \left(\frac{1}{2}U^2{}_x\right) = 0$$

which is exactly (1.7), the inviscid Burgers Equation [1]. This equation represents one of the more quintessential scalar hyperbolic conservation laws, and serves as a simplified model for more robust fluid dynamics. This equation will be treated quite comprehensively in this thesis. To fit all of the pieces together, (1.17) involves a *scalar* manipulation of the quantity U, is considered *hyperbolic* because of the positive value of its discriminant or, equivalently, because it possesses real and distinct eigenvalues for its associated Jacobian matrix, and the underlying quantity U is *conserved* throughout its domain as its dynamics are entirely determined by the fluxing of its density across arbitrary intervals. Such is the elucidation of *scalar hyperbolic conservation laws*.

Now, the degree to which these definitions are illuminating can be tenuous, but the significance of these designations will become apparent insofar as they help select the most suitable approach to solving each equation in kind, where certain approaches can vary drastically from type to type. By

clearly identifying the structure of the partial differential equations at hand, certain methodologies are either perfectly valid or terribly equipped at offering solutions. In a later section it will be shown that hyperbolic conservation laws in particular offer strategic advantages when constructing numerical approximation techniques, in large part because of their reliance on fluxing terms which lend themselves nicely to finite difference schemes in general. In conjunction with this is the fact that the solutions to such conservation laws follow characteristic curves, and this allows for the speedy convergence of methods like fast sweeping techniques.

FINITE DIFFERENCE METHODS

Although there has been sizeable mention of finite difference methods at large, a more thorough discourse on the matter is warranted and presented here. Starting with the somewhat innocuous looking limit definition of the derivative

1.18) $$f'(x_0) = \lim_{h\to 0} \frac{f(x_0+h)-f(x_0)}{h}$$

it should be immediately clear that a computerized framework tasked with solving any of the equations listed above (1.4 – 1.7) will necessarily make use of a modified interpretation of differentiation when compared with (1.18). Concepts like *instantaneous rate of change* or infinitesimals, epitomized by the inclusion of the limit in the above form, are completely alien to finite computer architectures, and must be dispensed with in favor of more quantifiable definitions [3,4]. A more palatable formulation occurs with the removal of the limit, for sufficiently small h, at the cost of no longer upholding true equality.

1.19) $$f'(x_0) \cong \frac{f(x_0+h)-f(x_0)}{h}$$

To regain equality, we can construct a general form using the first Lagrange polynomial expansion of the function $f(x)$, which is then differentiated, for some $\xi(x)$ between x_0 and $x_0 + h$

1.20) $$f(x) = \frac{f(x_0)(x-x_0-h)}{-h} + \frac{f(x_0+h)(x-x_0)}{h} + \frac{(x-x_0)(x-x_0-h)}{2} f''(\xi(x))$$

1.21) $$f'(x) = \frac{f(x_0+h)-f(x_0)}{h} + \frac{2(x-x_0)-h}{2} f''\big(\xi(x)\big) + \frac{(x-x_0)(x-x_0-h)}{2} f''(\xi(x))_x$$

If the value x_0 is set equal to x, then the more problematic terms in (1.21) drop out, and what is left is a very similar formulation to (1.19) but without recourse to approximation

1.22) $$f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{h}{2} f''(\xi)$$

Acknowledging that the magnitude of the error is directly proportional to the size of h, in the sense that previously the ability to ensure equality relied on the reduction of h to a magnitude of zero, a truncation error is thus incurred. Now, if we restrict ourselves to discussions of one variable x, and we replace h with its equivalent form as dx which is finite in value (in contrast to its otherwise infinitesimal connotation), we can succinctly rewrite (1.22) with our truncation error in big O notation, as a means of treating the incurred error as a bulk term

1.23) $$f'(x) = \frac{f(x+dx)-f(x)}{dx} - O(dx)$$

We can now apply (1.23) to a test case involving the advection equation (1.4), where *c* is taken to be negative unity. With the introduction of a second variable t, we also define the equivalent temporal discretization dt, along with sub/superscripts indicating the spatial and time nodes respectively, meaning that given an evaluation of U at time step n and spatial node i we can progress our evolution of U one step further in time to the $n+1^{th}$ time position, employing spatial values taken at i and/or its neighbors.

1.24) $$\frac{U_i^{n+1}-U_i^n}{dt} = \frac{U_{i+1}^n-U_i^n}{dx} + O(dx)$$ [Forward Difference]

1.25) $$\frac{U_i^{n+1}-U_i^n}{dt} = \frac{U_i^n-U_{i-1}^n}{dx} + O(dx)$$ [Backward Difference]

1.26) $$\frac{U_i^{n+1}-U_i^n}{dt} = \frac{U_{i+1}^n-U_{i-1}^n}{2dx} + O(dx^2)$$ [Centered Difference]

Taking this last equation (1.26), we can rearrange the terms to provide a direct evaluation of U at the next time step $n+1$ at the spatial node i

1.27) $$U_i^{n+1} = \frac{1}{2}\frac{dt}{dx}(U_{i+1}^n - U_{i-1}^n) + U_i^n + O(dx^2)$$

Thus, given initial and boundary conditions along some domain, we can march our solution forward in time by steps of dt for each iteration. To showcase a wider range of possible finite difference schemes, three-point and five-point schemes are presented below, in order to cultivate an appreciation for how each scheme in principle relies on a particular choice of stencil.

1.28) $$f'(x) = \frac{1}{2}\frac{1}{dx}\big(-3f(x) + 4f(x+dx) - f(x+2dx)\big) + O(dx^2)$$

[Three-Point Endpoint]

1.29) $$f'(x) = \frac{1}{12}\frac{1}{dx}(-25f(x) + 48f(x+dx) - 36f(x+2dx) + 16f(x+3dx) - \dots 3f(x+4dx)) + O(dx^4)$$

[Five-Point Endpoint]

And ad infinitum. Now two things to observe from (1.28) and (1.29) is that for one, as we increase the number of points which we include in our stencil (for example in (1.28), our stencil is considered to be the points j, $j+1$, and $j+2$, in contrast to the stencils of equations (1.24)-(1.26) which utilized two point stencils only), we are able to obtain higher and higher order accurate approximations to the actual value, which is seen by the degree of the truncation error term in each equation. This should make intuitive sense, as we are gathering more and more information about the underlying function U, and this will help to reveal the nature of its derivatives the more information we include. Second, it may have been guessed at, but a soft requirement for these methods is that the function U is continuous and well-defined on the entire domain. If a discontinuity were to arise which is spanned by any given choice of spatial stencil—a problem that is more apparent as we make use of more and more points per stencil—then our scheme will fail to converge to the proper value. If the difference between consecutive stencil nodes is substantially large, then an erroneous value for the derivative will result, as opposed to a piecewise solution. It would appear that an adaptive, many-point stencil scheme is needed to provide uniformly high order accurate approximations to piecewise discontinuous solutions to such partial differential equations, and that scheme is WENO.

2. WEIGHTED ESSENTIALLY NON-OSCILLATORY (WENO) FORMALIZATION

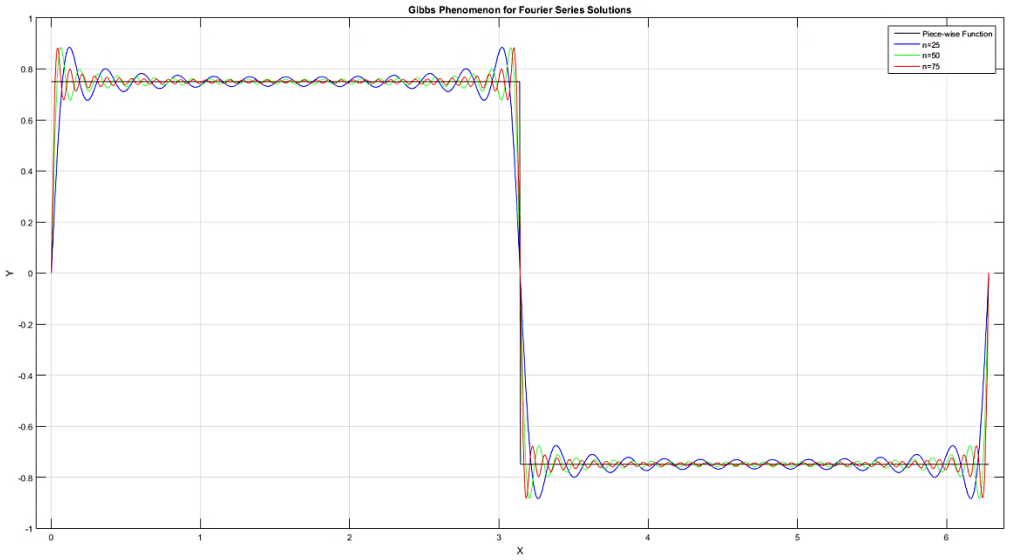
DIVERGENCE AND THE GIBB'S PHENOMENON

At present, we find ourselves confronted by a particular family of partial differential equations, namely scalar hyperbolic conservation laws epitomized by (1.17), while simultaneously we're equipped with a methodology for approximating solutions to such equations, here referring to finite difference schemes. Given smooth, continuous definitions for these equations and their solutions in general, there are no existential issues concerning the utilization of any given finite difference scheme. This is not the case however when functional discontinuities arise in either the partial differential equation's initial conditions or in the solutions themselves. The mere presence of such discontinuities poses a serious hindrance to any given finite difference framework's ability to approximate solutions, whereby convergence is highly jeopardized, and even when convergence is achieved, approximations can be riddled with errant oscillations known as the Gibb's Phenomenon in the direct vicinity surrounding these discontinuities [4]. Although convergence can sometimes be forced at the expense of a recklessly small Courant-Friedrichs-Lewy (CFL) condition—the ratio of temporal to spatial discretization—

2.1) $$\alpha \frac{dt}{dx} \leq CFL \qquad\qquad \alpha = max|f'(U)_x|$$

it is generally seen as a mitigation technique and not an asset to demand smaller and smaller time steps in order to salvage an improper algorithm. Such a small designation for the CFL condition where $dt \ll dx$ directly impacts the rate of convergence for these algorithms (a smaller CFL is slower by definition), and micromanaging this value is indicative of a structural algorithmic issue. For most scenarios a CFL condition between 0.5 and 1 is quite standard, where the more stable an algorithm is the larger its allowed CFL condition can be.

Now even if we were not to mind the hit to our running time and the overall increase in the iteration count for our approximation algorithm that would be guaranteed by a contrived choice of CFL condition, our forced convergence is still not without its drawbacks. When using finite sums of continuous functions to approximate discontinuous solutions, oscillatory behavior manifests itself at these points of discontinuity, and is apparent even for high order approximations. This is a notorious



[FIGURE 2]

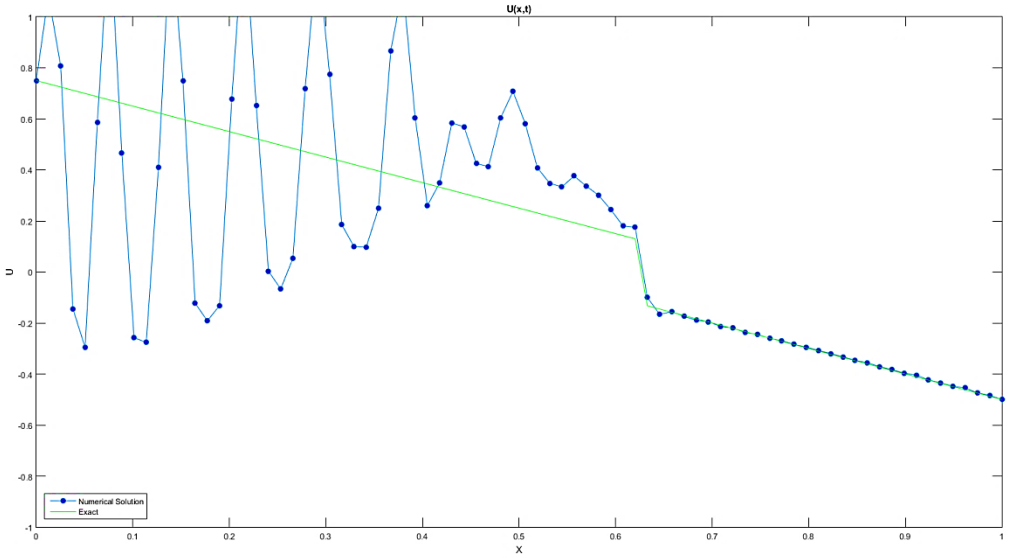
An exemplification of the Gibbs Phenomenon. A test function with its Fourier Series (2.2a/2.2b) can be approximated by its Fourier Series decomposition, and truncated versions of this series are graphed. The oscillations around the endpoints and central discontinuity are observed, highlighting the issues which arise as higher and higher order approximations are applied. Away from these discontinuities, the series attains high accuracy, but near such discontinuities large oscillations result.

2.2a) $$f(x) = \begin{cases} 0.75, & 0 < x < \pi \\ -0.75, & \pi \leq x < 2\pi \end{cases}$$

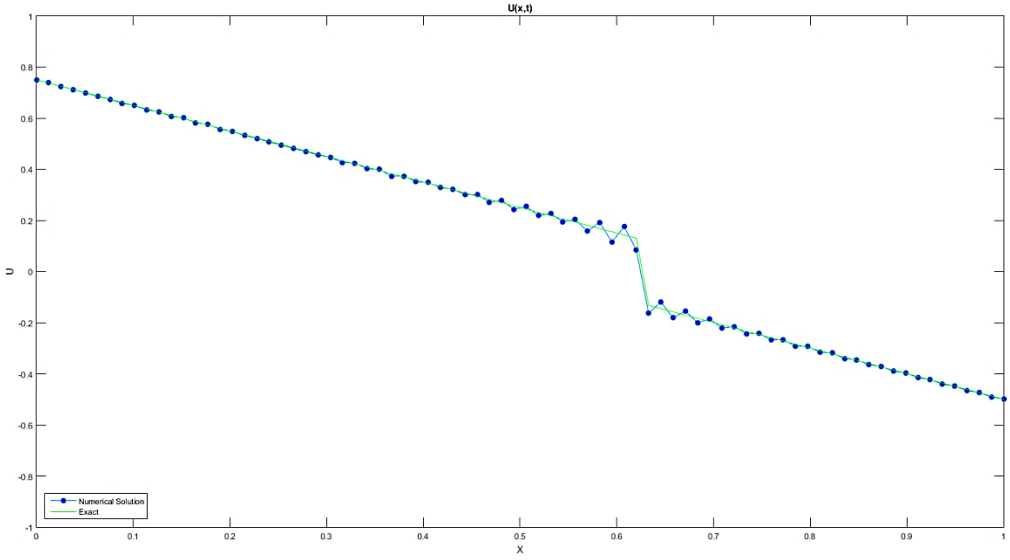
2.2b) $$f(x) = \frac{1.5}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} (1 - (-1^n)) \sin(nx)$$

affliction in the realm of Fourier Series (Figure 2), and this Gibb's Phenomenon also occurs in the utilization of finite difference methods.

To showcase this phenomenon as it pertains to the usage of finite difference methods, as well as to exemplify the inability for finite difference methods to converge (where even the best case scenario for convergence entails heavy oscillations) unless dramatically small CFL conditions are imposed, we briefly turn our attention to an example that will receive more rigorous treatment in sections 3 and 4. A centered difference scheme (1.26) is employed to solve (2.3a) with given boundary and initial conditions (2.3b) and steady state solution (2.3c), with a discretization of 80 spatial nodes across the domain.



[FIGURE 3]



[FIGURE 4]

Centered difference approximations to (2.3a) at $t = 5$ seconds with $N = 80$ evaluated with two separate CFL conditions. Above (Figure 3) the CFL condition is set to 0.15, where even such a small CFL condition is clearly divergent. Below (Figure 4) the CFL condition is set to 0.0015 which is comically small. Note the oscillations that are present, most notable around the discontinuity at $x = 0.625$

2.3a) $$U_t - \left(\frac{1}{2}U^2\right)_x = U \qquad 0 \le x \le 1$$

Boundary conditions

2.3b) $$U(0,t) = \frac{3}{4}, \quad U(1,t) = -\frac{1}{2}$$

Initial conditions

2.3b) $$U(x,0) = \begin{cases} \frac{3}{4}, & x < 0.5 \\ -\frac{1}{2}, & x \ge 0.5 \end{cases}$$

Steady state solution

2.3c) $$U(x) = \begin{cases} \frac{3}{4} - x, & x < 0.625 \\ x - \frac{1}{2}, & x \ge 0.625 \end{cases}$$

Due to the degree to which steady state convergence is impossible given the continual oscillations, the algorithm is stopped after $t = 5$ seconds.

What is ostensibly happening in (Figure 4) is that the problems (starkly visible on the left of Figure 3) associated with using a single stencil to approximate a discontinuous solution are mitigated by slowing down the progression of the solution with the implementation of a much smaller time step relative to the spatial step size, resulting in a CFL condition roughly 10 times smaller than the literature standard. In this way erroneous values attained near a discontinuity are not allowed to spread too quickly into the smoother regions, and these values are thus tempered as they dissipate. The Gibb's Phenomenon is quite apparent still in (Figure 4), and in (Figure 3) it can be seen that massive oscillations propagating from the central discontinuity are resulting in a chaotic scene at the leftmost endpoint. The key component to understand here is that the choice of stencil (1.26) at least twice per iteration (depending on the discretization) is forced to process functional data on either side of the discontinuity at $x = 0.625$. This results in the imposition of a large and inaccurate rate of change that is assumed to connect to the surrounding domain continuously, when instead it is a piecewise connection, and the battle of fitting a continuous interpretation to a piecewise reality is one of overshooting, then undershooting, then overshooting again etc., leading to ever present and intractable oscillations. Clearly such methods, as they currently stand, are unstable tools in approximating steady state solutions to hyperbolic conservation laws. In order to salvage the finite difference framework in light of the demands

inherent in discontinuous solutions, a dynamic and multi-stencil approach is to be developed in the proceeding section, one that is (essentially) free from the Gibb's Phenomenon and is convergent under more standard CFL conditions.

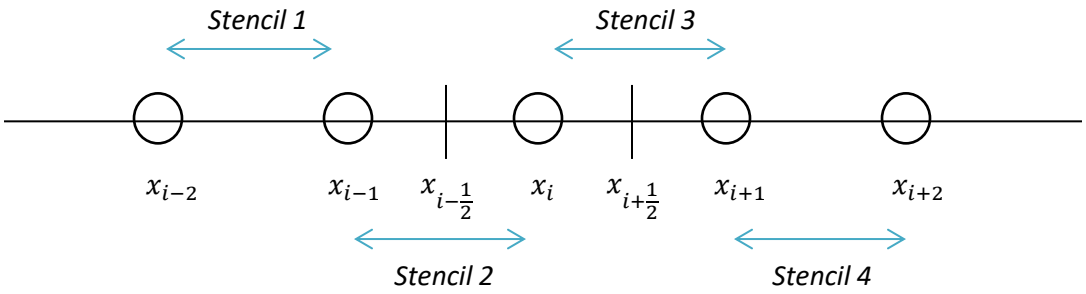
THIRD-ORDER WEIGHTED ESSENTIALLY NON-OSCILLATORY (WENO) FINITE DIFFERENCE SCHEME

The reliance on a single stencil without consideration for the underlying smoothness of the function has been shown to be the primary culprit for the failings that were encountered in the previous section in the development of a finite difference methodology in which to approximate discontinuous solutions to partial differential equations. The standard finite difference framework is just far too rigid in its uniform application of its choice of stencil at each point, and this blind insistence on a singular stencil at best results in oscillatory behavior near discontinuities, or clear divergence at worst. The third order accurate weighted essentially non-oscillatory scheme elects to remedy this issue by taking the convex combination of a set of four approximations, incorporating a total of five points embedded within four separate two-point stencils, each of which is weighted by a smoothness indicator that determines the degree to which a shock or discontinuity is spanned by that particular stencil [5, 8, 9, 10]. When a discontinuity is spanned by a specific stencil, it is weighted to nearly zero, thus minimizing the impact of approximations gleaned from this stencil, and the remaining stencils which query point values over smoother domains are to be used instead. Thus from four candidate stencils there will be at least one which is guaranteed to provide meaningful approximations. These stencils are tasked with approximating inter-nodal flux values, namely at $x_{i+\frac{1}{2}}$ and $x_{i-\frac{1}{2}}$, which are then used to approximate the total fluxing at the point x_i.

It is precisely this multi-stencil approach that is then coupled with an astute procedure for generating nonlinear weight coefficients which lays the groundwork for the success of WENO. The four candidate stencils shown in (Figure 4) can be written as

2.4) $$S^{(1)} = \{x_{i-2}, x_{i-1}\} \qquad S^{(2)} = \{x_{i-1}, x_i\}$$

$$S^{(3)} = \{x_i, x_{i+1}\} \qquad S^{(4)} = \{x_{i+1}, x_{i+2}\}$$



[FIGURE 5]

Four candidate stencils spanning five point values. Two approximations are made at inter-nodal values $x_{i+\frac{1}{2}}$ and $x_{i-\frac{1}{2}}$ in order to estimate the total flux at the central point value x_i by use of a linear combination of all available stencils, weighted based upon the underlying smoothness of the function.

where the superscript indicates the specific candidate stencil from which the points are chosen. Stencils for $n = 1,2$ are used in the approximation of the flux at the location $x_{i-\frac{1}{2}}$ where for $n = 3,4$ we are able to approximate the flux at $x_{i+\frac{1}{2}}$.

Now, much has been said of this concept of fluxing, and it is a key component to advection dominated conservation laws and is in need of some illumination. As exemplified in (Figure 1), the dynamics of some conserved quantity which is within a bounded region can be approximated by examining the degree to which this conserved quantity enters or leaves this region: a.k.a. its flux. There are several different monotone fluxing models available, however this thesis utilizes the Lax-Friedrichs flux splitting model exclusively [5, 16], given below

2.5a) $$\begin{cases} f^+(U_i) = \frac{1}{2}(f(U_i) + \alpha U_i) \\ f^-(U_i) = \frac{1}{2}(f(U_i) - \alpha U_i) \end{cases}$$ Lax-Friedrichs flux splitting

2.5b) $$\alpha = max|f'(U)_x|$$

The $+$ and $-$ superscripts are meant to indicate which direction is more heavily biased, or rather from which direction more point values are taken, either from the left or from the right respectively, as a means to make explicit the concept of right travelling or left travelling flux (a little confusing, I know).

We are finally equipped with the proper terminology to express the approximations alluded to earlier regarding flux values at inter-nodal values.

2.6a) $$\hat{f}^{+}_{i+\frac{1}{2}} = \omega_1 \left[\frac{1}{2}f^{+}(U_i) + \frac{1}{2}f^{+}(U_{i+1})\right] + \omega_2 \left[-\frac{1}{2}f^{+}(U_{i-1}) + \frac{3}{2}f^{+}(U_i)\right]$$

2.6b) $$\hat{f}^{-}_{i+\frac{1}{2}} = \omega_3 \left[\frac{3}{2}f^{-}(U_{i+1}) - \frac{1}{2}f^{-}(U_{i+2})\right] + \omega_4 \left[\frac{1}{2}f^{-}(U_i) + \frac{1}{2}f^{-}(U_{i+1})\right]$$

2.6c) $$\hat{f}^{+}_{i-\frac{1}{2}} = \omega_5 \left[\frac{1}{2}f^{+}(U_{i-1}) + \frac{1}{2}f^{+}(U_i)\right] + \omega_6 \left[-\frac{1}{2}f^{+}(U_{i-2}) + \frac{3}{2}f^{+}(U_{i-1})\right]$$

2.6d) $$\hat{f}^{-}_{i-\frac{1}{2}} = \omega_7 \left[\frac{3}{2}f^{-}(U_i) - \frac{1}{2}f^{-}(U_{i+1})\right] + \omega_8 \left[\frac{1}{2}f^{-}(U_{i-1}) + \frac{1}{2}f^{-}(U_i)\right]$$

We can now see all four two-point stencils from (2.4) with non-linear weights ω_n , where each flux value is some convex combination of two separate stencils spanning a total of three points each. Without loss of generality, the determination of the nonlinear weights ω_1 and ω_2 are given below, where there is a straightforward extension to the other six values.

2.7a) $$\omega_1 = \frac{\alpha_1}{\alpha_1+\alpha_2} \qquad \omega_2 = \frac{\alpha_2}{\alpha_1+\alpha_2}$$

2.7b) $$\alpha_1 = \frac{2/3}{\epsilon+\beta_1^2} \qquad \alpha_2 = \frac{1/3}{\epsilon+\beta_2^2}$$

2.7c) $$\beta_1 = (f^{+}(U_{i+1}) - f^{+}(U_i))^2 \qquad \beta_2 = (f^{+}(U_i) - f^{+}(U_{i-1}))^2$$

2.7d) $$\epsilon = 10^{-6}$$

The final equation above (2.7d) is implanted in (2.7b) in order to avoid an invalid expression for the denominator. Now we can sum together our flux values from (2.6a-d) to get

2.8a) $$\hat{f}_{i+\frac{1}{2}} = \hat{f}^{+}_{i+\frac{1}{2}} + \hat{f}^{-}_{i+\frac{1}{2}}$$

2.8b) $$\hat{f}_{i-\frac{1}{2}} = \hat{f}^{+}_{i-\frac{1}{2}} + \hat{f}^{-}_{i-\frac{1}{2}}$$

An approximation to the conservative flux difference is now possible, taken as the difference between these flux values at inter-nodal locations divided by their separation, in much the same way as the finite difference scheme presented in (1.26), but this time the separation is spanned by a single dx.

2.9) $$f(U_i)_x = \frac{1}{dx}\left(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\right)$$

This is the penultimate step in our formulation. Refreshing ourselves of the main focus of attention from earlier (1.1), written slightly differently in a fluxing model, we finally have

2.10) $$U(x_i)_t = -\frac{1}{dx}\left(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\right) + \phi_i$$

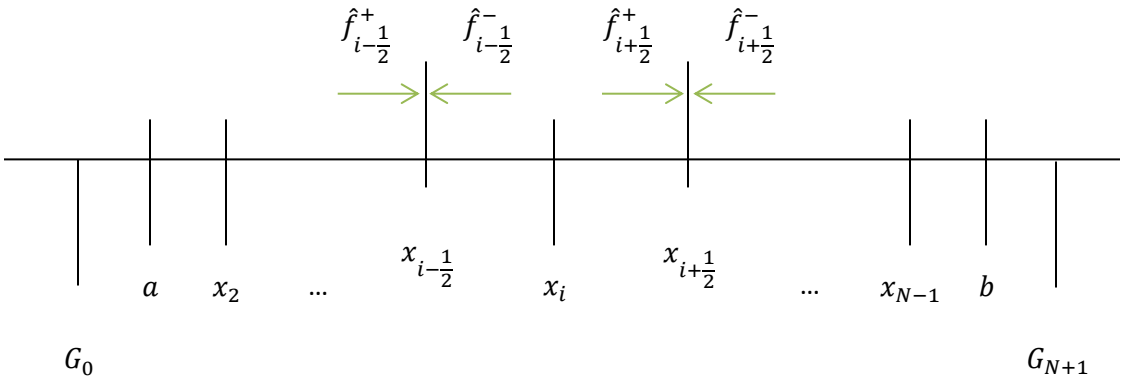
or written in a more discrete finite difference framework

2.11) $$\frac{U_i^{n+1} - U_i^n}{dt} = -\frac{1}{dx}\left(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\right) + \phi_i$$

with an explicit formulation for the next time step value

2.12) $$U_i^{n+1} = -\frac{dt}{dx}\left(\left(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\right) + \phi_i\right) + U_i^n$$

When there are explicitly defined values at the boundary for the function $U(x = a, x = b)$ given by the specifications of each partial differential equation, the full domain over which our finite difference scheme is concerned is actually the set $\{x_2, x_3, \dots, x_{N-2}, x_{N-1}\}$, as the values at the end point are held fixed and do not evolve in time, removing $\{x_1\}$ and $\{x_N\}$ from consideration. In order to implement the full WENO scheme at these new endpoints (namely $\{x_1\}$ and $\{x_{N-1}\}$) we still need two points on either side, as shown in (Figure 5). This is accomplished by adding ghost points values, established with the same regularity at a separation distance dx, indicated in (Figure 5) by G_k. These values are determined



[FIGURE 6]

WENO-3 discretization globally and locally about the point x_i. The conservative flux difference is the sum of the inflow and outflow at the inter-nodal point $x_{i+\frac{1}{2}}$ minus the sum of the inflow and outflow at the inter-nodal point $x_{i-\frac{1}{2}}$, diagrammatically displayed in (Figure 1), divided by their separation, which corresponds to (2.9). On the interval $[a, b]$ spanned by the points $\{x_1 = a, x_2, \dots, x_{N-1}, x_N = b\}$, there is the requirement to append to either end so-called ghost point values, denoted above by G_k.

by the steady state solution to the function $U(x)$ and are used as a means to maintain consistency with respect to the global utilization of the stencils outlined in (2.4). For particular examples where there is no fixed value for the boundary (see example 1) then a total of four ghost points are needed, two on either end of the domain. A single iteration is counted when an entire progression of all non-ghost and non-boundary (when fixed) values are made onto the next time step.

There is then a straightforward extension to functions of two variables, where the same adaptive stencil technique is applied to estimate flux values at inter-nodal points along each dimension independently, according to the two functions of $f(x)$ and $g(y)$ which define this fluxing. Given the two dimensional analogue to (1.1)

2.13) $$U_t + f(U)_x + g(U)_y = \phi$$

there is a similar modification to (1.40)

2.14) $$\frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{dt} = -\frac{1}{dx}\left(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\right) - \frac{1}{dy}\left(\hat{g}_{i+\frac{1}{2}} - \hat{g}_{i-\frac{1}{2}}\right) + \phi_{i,j}$$

where a different flux splitting calculation is made for the g function, this time given a subscript to avoid ambiguity

2.15a) $$\begin{cases} g^{+}(U_i) = \frac{1}{2}(g(U_i) + \alpha_y U_i) \\ g^{-}(U_i) = \frac{1}{2}(g(U_i) - \alpha_y U_i) \end{cases}$$

2.15b) $$\alpha_y = max\left|g'(U)_y\right|$$

and the rest of (2.6) to (2.9) can be followed without confusion for $g(y)$.

We end this section with a compact notation for the WENO methodology which will be used in the following sections when developing two separate total variation diminishing schemes. The right hand side of (2.14) can be subsumed within a single function and written as

2.16a) $$L\left(U_{i-r,j}, \dots, U_{i+s,j} : U_{i,j} : U_{i,j-r}, \dots, U_{i,j+s}\right) = -\frac{1}{dx}\left(\hat{f}_{i+\frac{1}{2}} - \hat{f}_{i-\frac{1}{2}}\right) - \frac{1}{dy}\left(\hat{g}_{i+\frac{1}{2}} - \hat{g}_{i-\frac{1}{2}}\right) + \phi_{i,j}$$

2.16b) $$\frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{dt} = L(U_{i-r,j}, \dots, U_{i+s,j} : U_{i,j} : U_{i,j-r}, \dots, U_{i,j+s})$$

2.16c) $$i = 1, \dots, N : j = 1, \dots, M$$

In (2.16a) the values for r and s indicate the rightmost and leftmost extension used in each stencil (i.e. each evaluation of the point x_i includes two points to the right and two points to the left), where the subscripts i and j indicate each of the two variables in (2.13). To ensure third order accuracy, we require at least the following holds, namely that for

2.17) $$r = s = 2 \qquad\qquad k = r + s - 1$$

where k denotes the degree of the specific finite difference scheme, of which here is presented a 3rd order scheme, apparent from (Figure 4).

Computational schemes which propose a criterion for convergence to be the attainment of successive iterations whose variations fall below some predetermined threshold are called total variation diminishing schemes [13]. These types of schemes analyze the entire functional domain and compare the most recent iteration to the previous iteration, in compliance with the following

2.18) $$\frac{1}{N}\sum_{i=1}^{N}\left|U_i^n - U_i^{n+1}\right| \leq \delta \qquad \delta = 10^{-11}$$

where n is in reference to the n^{th} time step, and the value in (2.18) is synonymous with the L^1 convergence error, which is the sum of the differences between the two iterations. Such a convergence criterion is impossible in the presence of oscillatory phenomena, where continual fluctuations will cause a ceaseless back-and-forth between each subsequent iteration. The successful implementation of this type of scheme is clear evidence that such oscillations (namely from the Gibbs phenomenon) are heavily minimized, directly proportional to the size of the variation limit, given by δ, and that global and uniform convergence has been achieved.

For the third-order Runge-Kutta scheme there are two successive refinements of the value $U(x_i, y_j)^{(1)}$ and $U(x_i, y_j)^{(2)}$ leading to the development of a third and final formulation of $U(x_i, y_j)^{n+1}$ as outlined below, fully expanded in multivariate notation

2.19a) $$U\left(x_i, y_j\right)^{(1)} =$$

$$U\left(x_i, y_j\right)^{n} + dt \cdot L\left(U\left(x_{i-r}, y_j\right)^{n}, \dots, U\left(x_{i+s}, y_j\right)^{n} : U\left(x_i, y_j\right)^{n} : U\left(x_i, y_{j-r}\right)^{n}, \dots, U\left(x_i, y_{j+s}\right)^{n}\right)$$

2.19b) $$U\left(x_i, y_j\right)^{(2)} =$$

$$\frac{3}{4}U\left(x_i, y_j\right)^{n} + \frac{1}{4}U(x_i, y_j)^{(1)} + \frac{1}{4}dt \cdot L\left(U\left(x_{i-r}, y_j\right)^{(1)}, \dots, U\left(x_{i+s}, y_j\right)^{(1)} : U\left(x_i, y_j\right)^{(1)} : U\left(x_i, y_{j-r}\right)^{(1)}, \dots, U\left(x_i, y_{j+s}\right)^{(1)}\right)$$

2.19c) $$U(x_i, y_j)^{n+1} = \frac{1}{3}U(x_i, y_j)^{n} + \frac{2}{3}U(x_i, y_j)^{(2)} + \frac{2}{3}dt \cdot L\left(U(x_{i-r}, y_j)^{(2)}, \dots, U(x_{i+s}, y_j)^{(2)} : U(x_i, y_j)^{(2)} : U(x_i, y_{j-r})^{(2)}, \dots, U(x_i, y_{j+s})^{(2)}\right)$$

2.19d) $$\alpha \frac{dt}{dx} \leq CFL = 0.5 \qquad dt = \frac{1}{2\alpha}dx$$

2.19e) $$\alpha = max\left(|f'(U)|_x \; or \; |g'(U)_y|\right)$$

Written in more succinct notation

2.20a) $$U^{(1)} = U^n + dt \cdot L(U^n)$$

2.20b) $$U^{(2)} = \frac{3}{4}U^n + \frac{1}{4}U^{(1)} + \frac{1}{4}dt \cdot L(U^{(1)})$$

2.20c) $$U^{n+1} = \frac{1}{3}U^n + \frac{2}{3}U^{(2)} + \frac{2}{3}dt \cdot L(U^{(2)})$$

In the literature [9] the generalized left hand side of (2.20a-c) U^k is often written as $\overline{U}^k$ in order to concretize the fact that these values are mere continual approximations, but this has been omitted here to reduce the avalanche of notation to a deluge instead. In the case of scalar hyperbolic conservation laws involving a source term that is a function of time (namely for partial differential equations for which a steady state solution is not possible [of which only example one is of this type]), the function L is a function of x, y, and t, requiring modifications to (2.20a-c) in the following manner

2.21a) $$U^{(1)} = U^n + dt \cdot L(U^n, t^n)$$

2.21b) $$U^{(2)} = \frac{3}{4}U^n + \frac{1}{4}U^{(1)} + \frac{1}{4}dt \cdot L(U^{(1)}, t^n + dt)$$

2.21c) $$U^{n+1} = \frac{1}{3}U^n + \frac{2}{3}U^{(2)} + \frac{2}{3}dt \cdot L(U^{(2)}, t^n + \frac{1}{2}dt)$$

To reiterate, the parenthesized superscripts are in reference to the intermediate refinements of the approximation to U^{n+1}, which are then used in a linear combination with the original value of U^n to approximate the next time step. Also in the above formulations (2.20/2.21), reference to the specific point locations are omitted for the sake of brevity, however they are fully detailed in (2.19). Now it is this scheme that will be used in the time marching approach that will be used as a comparison to the fast sweeping Gauss-Seidel Successive Over-Relaxation scheme.

FAST-SWEEPING SUCCESSIVE OVER-RELAXATION GAUSS-SEIDEL

In much the same fashion as the previous formulation, the fast-sweeping Gauss-Seidel scheme is defined by two consecutive refinements of the approximation to $U(x_i, y_j)^{n+1}$ ending in a third and final approximation which interweaves the latest refinement $U(x_i, y_j)^{(2)}$. There are three key additional concepts which this scheme employs, namely the incorporation of the most up-to-date values possible, the weighted averaging of this latest information with the current value, and the utilization of alternating sweeping directions. On the first of these regarding the most recent data values, this means that as we continue through the domain, whenever possible we make use of the latest iterated value, which may be from the currently-being-calculated time step (i.e. the $n+1^{\text{th}}$ time step in the calculation of the $n+1^{\text{th}}$ time step at a different point, see [Figure 6]).

2.22a) $$U_{i,j}^{(1)} = U_{i,j}^{n} + \frac{\gamma}{\frac{\alpha_x}{dx}+\frac{\alpha_y}{dy}} \cdot L\left(U_{i-r,j}^{*}, \dots, U_{i+s,j}^{*} : U_{i,j}^{n} : U_{i,j-r}^{*}, \dots, U_{i,j+r}^{*}\right)$$

2.22b) $$U_{i,j}^{(2)} = U_{i,j}^{(1)} + \frac{1}{4}\frac{\gamma}{\frac{\alpha_x}{dx}+\frac{\alpha_y}{dy}} \cdot L\left(U_{i-r,j}^{**}, \dots, U_{i+s,j}^{**} : U_{i,j}^{(1)} : U_{i,j-r}^{**}, \dots, U_{i,j+r}^{**}\right)$$

2.22c) $$U_{i,j}^{n+1} = U_{i,j}^{(2)} + \frac{2}{3}\frac{\gamma}{\frac{\alpha_x}{dx}+\frac{\alpha_y}{dy}} \cdot L\left(U_{i-r,j}^{***}, \dots, U_{i+s,j}^{***} : U_{i,j}^{(2)} : U_{i,j-r}^{***}, \dots, U_{i,j+r}^{***}\right)$$

To clarify, the constant γ is the same value obtained from (2.19) which is the CFL condition, here set to 0.5. Also, the asterisk superscripts for the function U are meant to indicate the "to be determined" nature of these values, in the sense that when unavailable, the previous refinement's or time step's

value is to be used. However if a newer value has been calculated and is available within the current stencil consideration, then it is used instead, coming from either the next refinement or time step.

2.23a) $$U^{*} = \begin{cases} U_{i,j}^{n} \; otherwise \\ U_{i,j}^{(1)} \; when \; available \end{cases}$$

2.23b) $$U^{**} = \begin{cases} U_{i,j}^{(1)} otherwise \\ U_{i,j}^{(2)} \; when \; available \end{cases}$$

2.23c) $$U^{***} = \begin{cases} U_{i,j}^{(2)} \; otherwise \\ U_{i,j}^{n+1} \; when \; available \end{cases}$$

As evidenced in (Figure 6), when there is the availability of new information, changes can be made to the existing stencil parameters such that this most recent data occupies the place of now essentially overwritten data. In contrast, the Runge-Kutta scheme previously outlined will perform an un-adjusted evolution of all points within a domain, sticking to the same stencil which takes only data from a predetermined time step or refinement. From (2.23) it is clear that there are two distinct values which can be used in calculating either $U^{(1)}$, $U^{(2)}$, or U^{n+1}, and therefore any particular U^{*}, U^{**}, or U^{***} is to be determined dynamically, based upon what is currently available. The convention in (2.23) is that the top value is the default value, and the bottom value is to be used when it first becomes available, exactly as what is diagrammatically implied in (Figure 6).

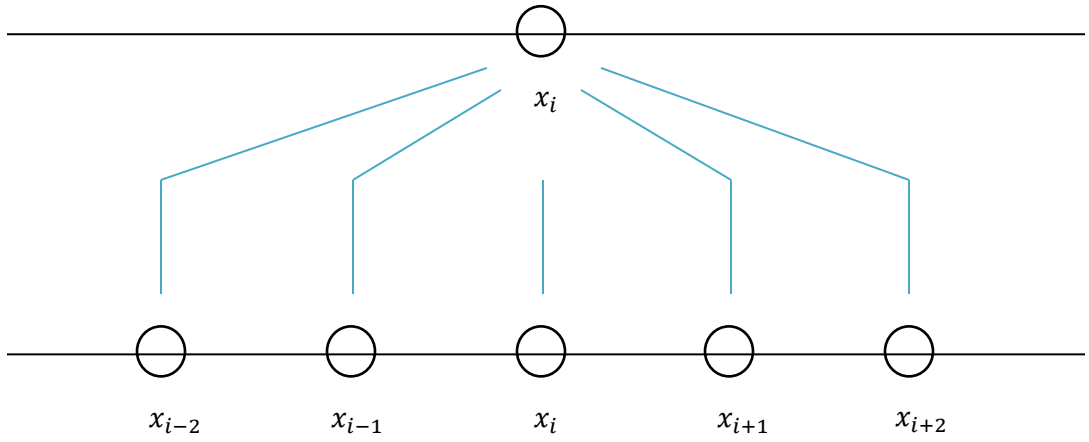
On the second of the aforementioned three key components, the concept of incorporating the newest available information is further developed in the Gauss-Seidel framework by also allowing for a weighted average of current and previous values to be calculated. This weighted average is moderated by a relaxation parameter ω which can be tailored to fine tune computational performance, an important aspect that will receive extensive attention in later examples.

2.24) $$U^{(k)} = \omega \overline{U}^{(k)} + (1 - \omega) U^{(k-1)}$$

Time-step

Iteration 1

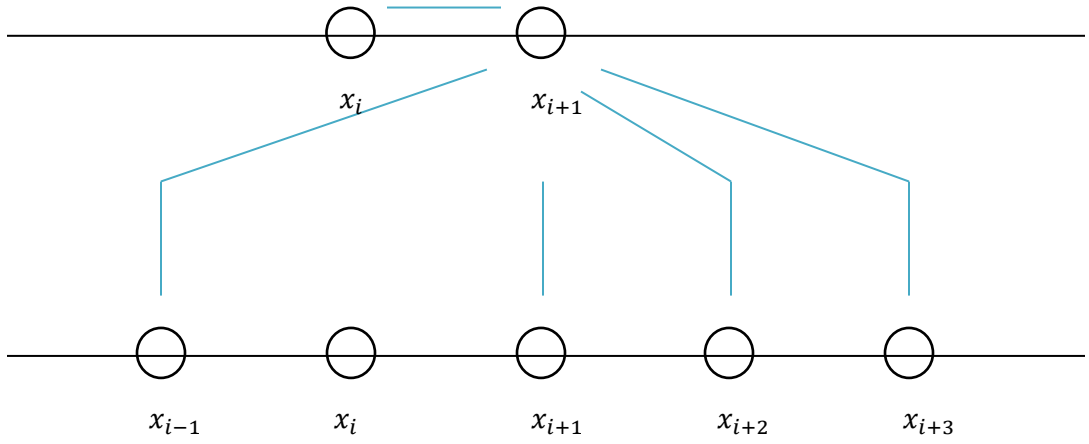
$n+1$



Time-step

Iteration 2

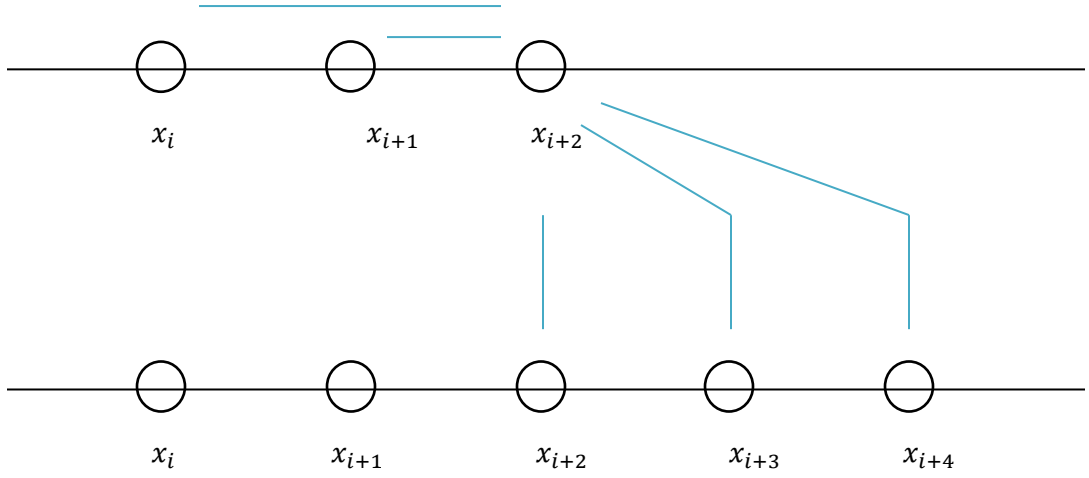
$n+1$



Time-step

Iteration 3

$n+1$



[FIGURE 7]

Three successive iterations progressing the points x_i, x_{i+1}, and x_{i+2} from time step n to $n+1$. This is an oversimplified diagrammatic showcasing of the discretionary nature of the Gauss-Seidel scheme, where once a new value has been calculated, it can be substituted in the place of the previous time step's (or previous refinement's) value, allowing the parameters of the stencil to adjust itself in favor of incorporating the most recently available values.

In a generalized form (2.24) displays how either a refinement or a next time step value for some $U^{(k)}$ can be obtained through a process of averaging the most recent Gauss-Seidel iteration $\overline{U}^{(k)}$ (given a bar to indicate its stature as an intermediate value) with the previously determined $U^{(k-1)}$, based upon the value of the relaxation parameter ω. We then extend the definition from (2.22) to include this formulation involving such a relaxation parameter and provide a full algorithmic scheme

2.25a)
$$\begin{cases} \overline{U}_{i,j}^{(1)} = U_{i,j}^{n} + \frac{\gamma}{\frac{\alpha_x}{dx}+\frac{\alpha_y}{dy}} \cdot L\left(U_{i-r,j}^{*}, \dots, U_{i+s,j}^{*} : U_{i,j}^{n} : U_{i,j-r}^{*}, \dots, U_{i,j+r}^{*}\right) \\ U_{i,j}^{(1)} = \omega \overline{U}_{i,j}^{(1)} + (1-\omega) U_{i,j}^{n} \end{cases}$$

2.25b)
$$\begin{cases} \overline{U}_{i,j}^{(2)} = U_{i,j}^{(1)} + \frac{1}{4}\frac{\gamma}{\frac{\alpha_x}{dx}+\frac{\alpha_y}{dy}} \cdot L\left(U_{i-r,j}^{**}, \dots, U_{i+s,j}^{**} : U_{i,j}^{(1)} : U_{i,j-r}^{**}, \dots, U_{i,j+r}^{**}\right) \\ U_{i,j}^{(2)} = \omega \overline{U}_{i,j}^{(2)} + (1-\omega) U_{i,j}^{(1)} \end{cases}$$

2.25c)
$$\begin{cases} \overline{U}_{i,j}^{n+1} = U_{i,j}^{(2)} + \frac{2}{3}\frac{\gamma}{\frac{\alpha_x}{dx}+\frac{\alpha_y}{dy}} \cdot L\left(U_{i-r,j}^{***}, \dots, U_{i+s,j}^{***} : U_{i,j}^{(2)} : U_{i,j-r}^{***}, \dots, U_{i,j+r}^{***}\right) \\ U_{i,j}^{n+1} = \omega \overline{U}_{i,j}^{n+1} + (1-\omega) U_{i,j}^{(2)} \end{cases}$$

The third and final key component to the fast sweeping Gauss-Seidel scheme is the utilization of alternating sweeping directions along which the functional domain is scanned. For more standard implementations of finite difference schemes in general and for the the time-marching Runge-Kutta scheme in particular the convention is to start each iteration as a scan from one side to the other without adjustment, according to (2.16b). This is referred to as a time marching method which, like a typewriter, operates with a singular logic. By allowing for alternating sweeping directions, however, convergence to steady state solutions can be significantly sped up, and in some cases baseline convergence is achieved when a time marching method would result in divergence. For the two dimensional case there are four discrete possible sweeping directions, given below

2.26a) $x \in \{x_1 = a, x_2, x_3, \dots, x_i, \dots, x_{N-2}, x_{N-1}, x_N = b\}$

2.26b) $y \in \{y_1 = a, y_2, y_3, \dots, y_j, \dots, y_{M-2}, y_{M-1}, y_M = b\}$

2.26c) $1: i = 1:N, \quad j = 1:M$

$2: i = N:1, \quad j = 1:M$

$3: i = N:1, \quad j = M:1$

$4: i = 1:N, \quad j = M:1$

We can now revisit (Figure 6) and appreciate the fact that the diagram is a one dimensional depiction of the first type of sweep given in (2.26c), namely a left-to-right stencil progression along the domain. If this same figure were to showcase the second sweeping direction, a mirror image can be conjured up where points from the right are incorporated into the stencil parameters as it moves to the left. It should be noted that these alternating sweeping directions are strictly implemented per iteration, meaning that one set iteration is performed using one set sweeping direction, and only once every value has been progressed to the next time step will an alternation of sweeping direction occur.

3. NUMERICAL EXAMPLES

NOTES ON THE NUMERICAL EXAMPLES

For the implementation and analysis of the WENO scheme, there are several specifications to be listed here. Each example's approximation to the steady state solution is refined until successive iterations exhibit variations below a set threshold, as mentioned earlier and given by (2.18), with the exception of example 1 where the solution is progressed until $t = 1$, give-or-take some residual based upon the choice of dt. For example 1, a centered difference formulation (1.26) is compared with the time-marching Runge-Kutta scheme outlined in (2.21), which is the WENO framework which includes a treatment for time-dependent source terms. For all examples, the ghost points along the domain are a fixed distance of dx to the right and left of the endpoints $[a, b]$, and these values are set using the steady state solution values. The accuracy in terms of the L_1 and L_∞ norms are determined respectively

3.1a) $$L_1 = \frac{1}{N}\sum_{i=1}^{N}|\overline{U}_i^m - U_i^m|$$

3.1b) $$L_\infty = max|\overline{U}_i^m - U_i^m|$$

where $\overline{U}$ is the approximate solution (given a bar to avoid ambiguity), U is the exact solution, and the superscript versions $\overline{U}^m$ and U^m are to denote the fact that these are evaluations that do not consider some local domain surrounding the location of discontinuity (i.e. $\overline{U}^m = \overline{U}$ minus some fixed subset of points), in order for the accuracy of the approximation to be measured strictly in the smooth regions of approximation (Figure 9). Meaning if the location of discontinuity were given by c, then

3.2a) $$\overline{U}_m = \begin{cases} 0 & if\ |x_i - c| \le 0.1 \\ \overline{U} & otherwise \end{cases}$$

3.2b) $$U_m = \begin{cases} 0 & if\ |x_i - c| \le 0.1 \\ U & otherwise \end{cases}$$

Thus for all point values within this range of the discontinuity, the approximation and the exact solution are zeroed out, in order for the L_1 and L_∞ errors to be strict measurements of the convergence within the smooth regions of the domain.

The order of accuracy for both the L_1 and L_∞ norms is a relative value, which is why in the tabulations to follow the first row is left blank, and is defined below

3.3) $$order_k = \frac{\ln(\frac{previous\ L_k}{current\ L_k})}{\ln(2)}$$

where k is in reference to either the order of the L_1 or L_∞ norms respectively. The designation of 'previous' and 'current' is meant to denote the relative discretization used, which should be made clear from the given tables.

The iteration optimization percentage is a ratio of the quickest and slowest convergences across different extrapolation factors for each discretization, and this value is obtained by

3.4) $$\text{Iteration reduction } \% = \left(1 - \frac{min}{max}\right) \times 100\%$$

where the min value is taken when $\omega = 1$. Finally, in calculating the residual history throughout the approximation process for each example, the residual value at the n^{th} iteration is determined by

3.5) $$\overline{U}^n_{res} = max|\overline{U}^n - \overline{U}^{n-1}|$$

which is to say the residual is measured as the largest deviation from the previous approximation to the most current. This process clearly ends when the variations are less than 10^{-11} in magnitude, in keeping with (2.18).

CFL CONDITIONS

Several of the references [18, 19] discuss how the WENO framework allows for accurate approximations even when utilizing relatively large CFL conditions (greater than 0.5 in general, specifically larger than 1).

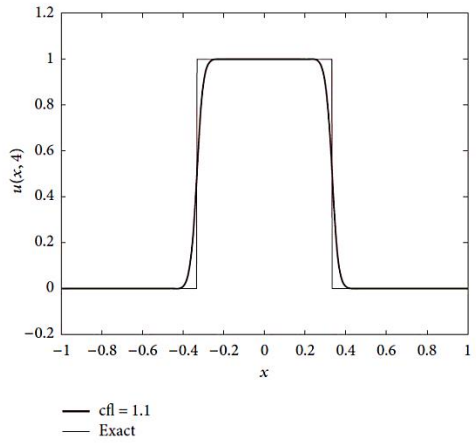


FIGURE 6: Results of Problem II at cfl number 1.1 with number of spatial steps, $N = 800$.

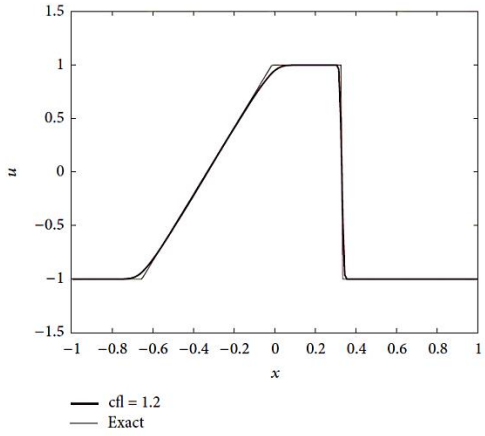
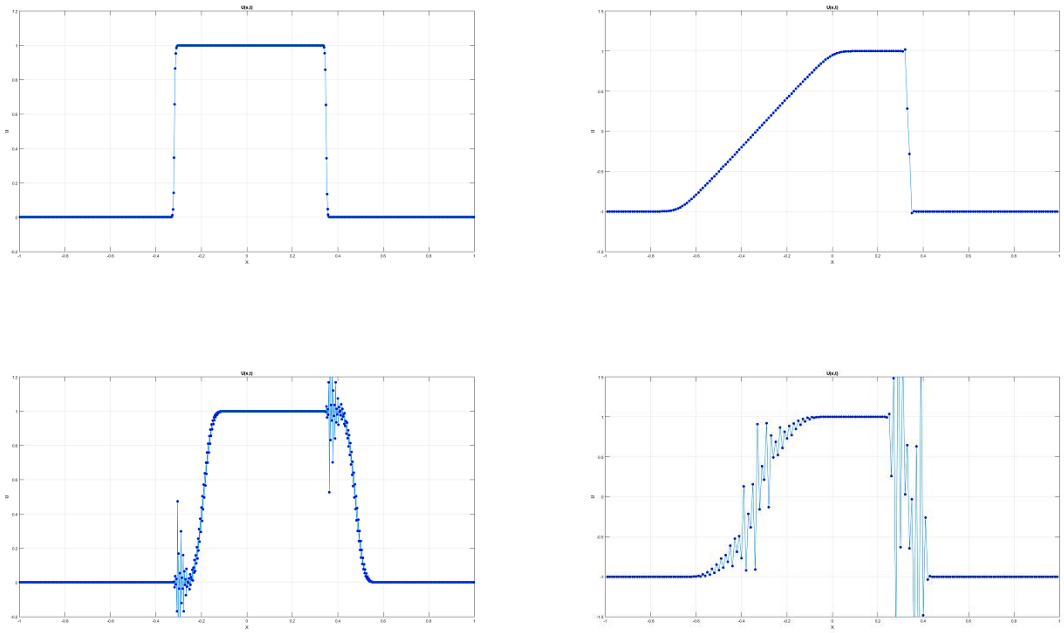


FIGURE 8: Solutions for Problem III with 200 cells at $T = 0.32$ using WENO3 scheme at cfl number 1.3.



[Figure 8]

Appadu, A.R. and Peer, A. A. I. "Optimized Weighted Essentially Non-Oscillatory Third-Order Schemes for Hyperbolic Conservation Laws." Journal of Applied Mathematics, Article ID 428681, 2013, page 11. The original reference and reproduction of the approximations from [19], where the lowest two images are the approximations at slightly larger CFL conditions, just as divergence results. These issues clearly arise at the locations of discontinuity.

Problem III

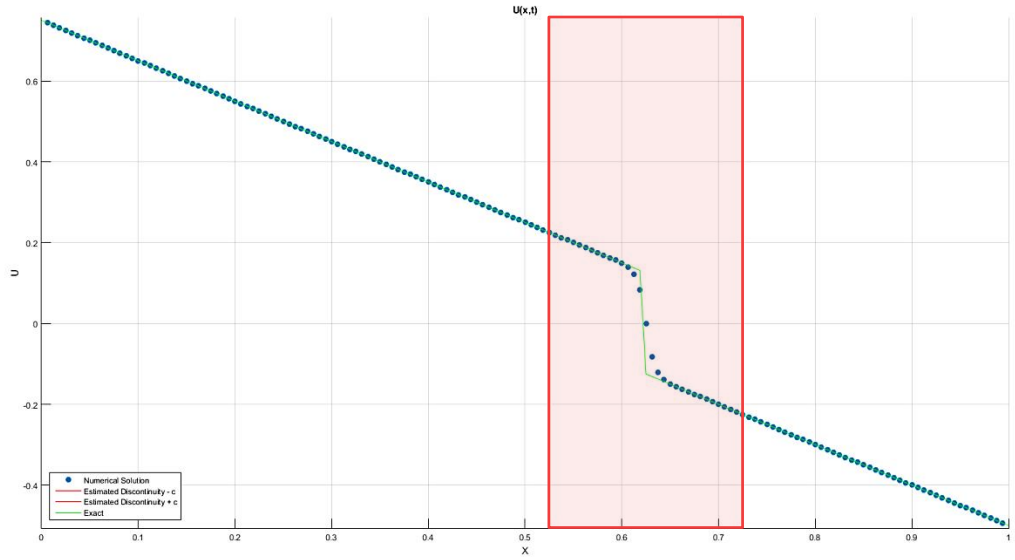
$$U_t + \left(\frac{1}{2}U^2\right)_x = 0$$

$$U(x,0) = \begin{cases} 1 & for\ |x| < \frac{1}{3} \\ -1 & otherwise \end{cases}$$

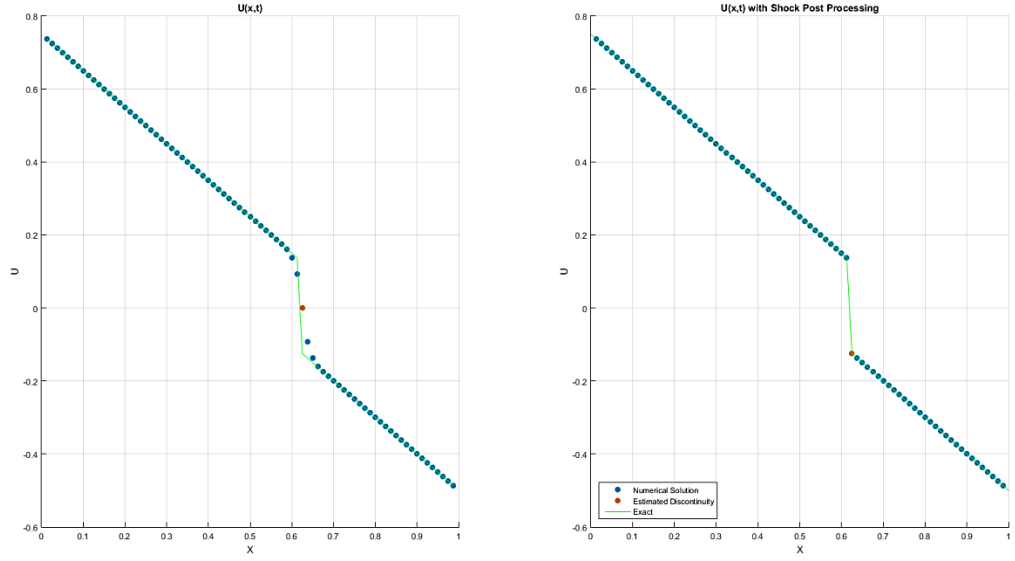
examining the iteration rates of time-marching and fast-sweeping algorithms, with additional refinements made to the Gauss-Seidel extrapolation factor. In Example 6 the CFL condition is increased to show further optimizations, and is then set too large to demonstrate how even a simple example can disintegrate at too large a CFL condition. In a sense you are rushing the solution more and more by demanding larger and larger CFL values.

SHOCK POST-PROCESSING

A third-order accurate approximation is only so good, and at locations of discontinuity there is a sizeable disparity between the exact solution and the estimate [7, 13]. So sizeable that it is standard practice to take error measurements from the smooth regions of the solution alone, disregarding a sub-interval surrounding the discontinuity where accuracy is not guaranteed. In an intuitive sense, it is precisely around discontinuities that we begin to limit how much information we have available to make approximations, for in these areas we have at least one stencil that is weighted closer to zero (the stencil bridging the discontinuity), and we are thus relying on less information, hence a lower order approximation. However, once steady state has been reached, I noticed that the sum of the smoothness indicators (2.7c) gave a very good approximation to areas of discontinuities, and in cases of a strong shock (focused on a single point), estimations to the exact location can be made. Given this information, we can enforce a left- or right-sided limit to either side of the shock point, and thus give better resolution around the discontinuity location. Therefore instead of removing an entire section of the domain (Figure 9), we can remove just a single point, the discontinuity itself, and linearly approximate (Example 2) and non-linearly second-order centered difference approximate (Example 3) the discontinuity region. The error in strictly the smooth regions is still superior (i.e. smaller), but improvements to the discontinuity region can be made with this post-processing approach. In (Table 1) the only excluded point is the estimated point of discontinuity, and so the error measurements are essentially across the entire domain (notice how much larger they are relative to the Table 1 values when the smooth regions alone are considered).



[FIGURE 9]



[FIGURE 10]

Example 2 with the excluded region (Figure 9) and the singular point of discontinuity given the pre- and post-processing procedure (Figure 10).

N	L_1 error	L_1 error Post Processing
40	0.0052	4.6372e-05
80	0.0030	2.9826e-05
160	0.0016	3.8209e-05
320	8.5088e-04	1.8379e-05
640	4.3751e-04	8.0143e-06

[TABLE 1]

A comparison of the L_1-$norm$ at different discretization and CFL settings for both the time-marching Runge-Kutta WENO and centered difference schemes for Example 2, where the error is measured across the entire domain, excluding the singular discontinuous point.

(Table 1) shows the potential for a post-processing scheme to reduce the error in the regions surrounding discontinuities, and that error measurements in the smooth region can be considered up until the discontinuity itself, assuming a strong shock is present. With just a linear interpolation for the neighboring points (here taken to be five points on either side of the estimated discontinuity), a reduction of two orders of magnitude is possible, with clearly room for further improvement. In contrast, all later examples will take error measurements in the smooth regions of the solution alone.

In real world scenarios such adherence to the piece-wise nature of the solution may be unrealistic, but in simple cases where assumptions of linearity can be made and the presence of strong shocks is known, such post-processing can boost accuracy in the discontinuous region. The smoothness indicator sum is just that, from (2.7c).

3.6a) $$B^n = \sum_{i=1}^{N} \sum_{k=1}^{8} \beta_k$$

3.6b) $$if\ max(B^n) = B_i^n \text{ then estimated discontinuity at } x_i$$

namely a double sum across all points and all smoothness indicator values. The maximum values of this crude function (XXX) are actually quite accurate at defining either a range or a point to the location of functional discontinuity. Once steady state has been reached, this value settles on the discontinuity region, and a post-processing procedure can be applied.

SCALAR HYPERBOLIC PARTIAL DIFFERENTIAL EQUATION WITH TIME-DEPENDENT SOURCE TERM

Example 1:

3.7a) $$U_t + \left(\frac{1}{2}U^2\right)_x = cos(x+t)[1+sin(x+t)] \qquad 0 \le x \le 2\pi$$

Initial conditions

3.7b) $$U(x,0) = sin(x)$$

Steady state

3.7c) $$U(x,t) = sin(x+t)$$

1-DIMENSIONAL NONLINEAR BURGER'S EQUATIONS WITH SOURCE TERMS

Example 2:

3.8a) $$U_t - \left(\frac{1}{2}U^2\right)_x = U \qquad 0 \le x \le 1$$

Boundary conditions

3.8b) $$U(0,t) = \frac{3}{4} \qquad U(1,t) = -\frac{1}{2}$$

Initial conditions

3.8c) $$U(x,0) = \begin{cases} \frac{3}{4}, & x < \frac{1}{2} \\ -\frac{1}{2}, & x \ge \frac{1}{2} \end{cases}$$

Steady state solution

3.8d) $$U(x) = \begin{cases} \frac{3}{4} - x, & x < 0.625 \\ -x + \frac{1}{2}, & x \ge 0.625 \end{cases}$$

Example 3:

3.9a) $U_t + \left(\frac{1}{2}U^2\right)_x = sin(x)\,cos(x)$ $0 \le x \le \pi$

Boundary conditions

3.9b) $U(0,t) = 0$ $U(\pi,t) = 0$

Initial conditions

3.9c) $U(x,0) = \frac{1}{2}sin(x)$

Steady state

3.9d) $$U(x) = \begin{cases} sin(x), & 0 \le x < \frac{2\pi}{3} \\ -sin(x), & \frac{2\pi}{3} \le x \le \pi \end{cases}$$

2-DIMENSIONAL NONLINEAR BURGER'S EQUATIONS WITHOUT AND WITH SOURCE TERMS

Example 4:

3.10a) $U_t + \left(\frac{1}{2}U^2\right)_x + U_y = 0$ $0 \le x, y \le 1$

Boundary conditions

3.10b) $$U(x,y) = \begin{cases} \frac{3}{2}, & x = 0 \\ -\frac{5}{2}x + \frac{3}{2}, & y = 0 \\ -1, & x = 1 \end{cases}$$

Initial conditions

3.10c) $U(x,y,0) = -\frac{5}{2}x + \frac{3}{2}$

Steady state solution

3.10d) $$U(x,y)=\begin{cases}\frac{3}{2}, & for\ x,y\ between\ l1\ and\ l3\\ -1, & for\ x,y\ between\ l2\ and\ l3\\ -\frac{5}{2}\left(\frac{3}{5}+\frac{\frac{2}{5}\left(x-\frac{3}{5}\right)}{\frac{2}{5}-y}\right)+\frac{3}{2}, & for\ x,y\ between\ l1\ and\ l2\end{cases}$$

Example 5:

3.11a) $$U_t+\left(\frac{U^2}{2\sqrt{2}}\right)_x+\left(\frac{U^2}{2\sqrt{2}}\right)_y=-U\pi\cos\left(\pi\frac{x+y}{\sqrt{2}}\right) \qquad 0\le x,y\le\frac{1}{\sqrt{2}}$$

Boundary conditions

3.11b) $$U(x,y,t)=\begin{cases}1-sin\left(\pi\frac{y}{\sqrt{2}}\right), & x=0,\frac{y}{\sqrt{2}}\le x_s\\ 1-sin\left(\pi\frac{x}{\sqrt{2}}\right), & y=0,\frac{x}{\sqrt{2}}\le x_s\\ -\frac{1}{10}-sin\left(\pi(\frac{1}{2}+\frac{y}{\sqrt{2}})\right)\\ -\frac{1}{10}-sin\left(\pi(\frac{1}{2}+\frac{x}{\sqrt{2}})\right), & x,y=\frac{1}{\sqrt{2}}\end{cases}$$

3.11c) $$x_s=0.1486$$

Initial conditions

3.11d) $$U(x,y,0)=\begin{cases}1, & 0\le\frac{x+y}{\sqrt{2}}\le\frac{1}{2}\\ -\frac{1}{10}, & \frac{1}{2}<\frac{x+y}{\sqrt{2}}\le 1\end{cases}$$

Steady state solution

3.11e) $$U(x,y)=\begin{cases}1-sin\left(\pi\frac{x+y}{\sqrt{2}}\right), & 0\le\frac{x+y}{\sqrt{2}}\le x_s\\ -\frac{1}{10}-sin\left(\pi\frac{x+y}{\sqrt{2}}\right), & x_s\le\frac{x+y}{\sqrt{2}}\le 1\end{cases}$$

Example 6:

3.12a) $U_t + \left(\frac{U^2}{2\sqrt{2}}\right)_x + \left(\frac{U^2}{2\sqrt{2}}\right)_y = sin\left(\frac{x+y}{\sqrt{2}}\right) cos\left(\frac{x+y}{\sqrt{2}}\right)$ $\frac{\pi}{4\sqrt{2}} \leq x, y \leq \frac{3\pi}{4\sqrt{2}}$

Boundary conditions

3.12b) $U(x,y) = sin(\frac{x+y}{\sqrt{2}})$ $x, y = \frac{\pi}{4\sqrt{2}}$ or $x, y = \frac{3\pi}{4\sqrt{2}}$

Initial conditions

3.12c) $U(x,y) = \beta sin\left(\frac{x+y}{\sqrt{2}}\right)$ $\beta = \frac{3}{2}$

Steady state solution

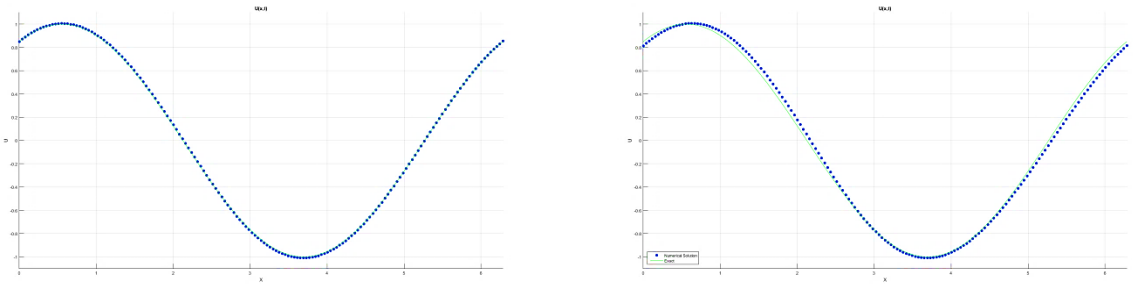
3.12d) $U(x,y) = sin\left(\frac{x+y}{\sqrt{2}}\right)$

4. MATHWORKS' MATLAB IMPLEMENTATION OF WENO-3 ALGORITHM

EXAMPLE SOLUTIONS

Example 1:

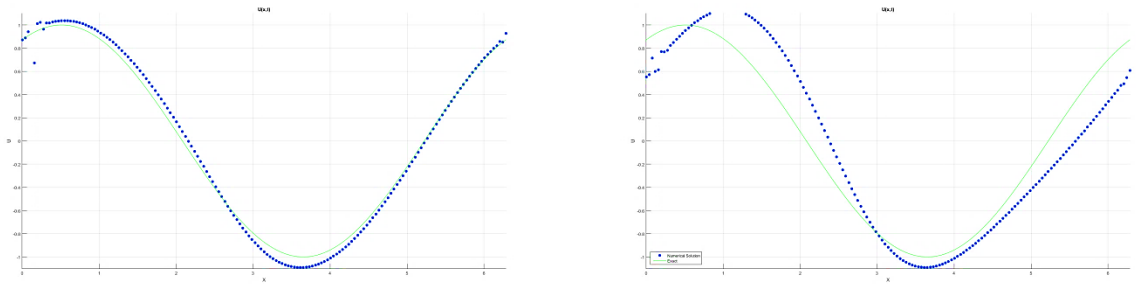
$$U_t + \left(\frac{1}{2}U^2\right)_x = cos(x+t)\,[1 + sin(x+t)]$$



[FIGURE 11]

WENO approximation (Figure 11 left) and centered difference approximation (Figure 11 right) to (3.7a) at $t = 1$ and $N = 160$. The CFL condition is set to 0.5.

Example 1 is meant purely to introduce the WENO methodology in comparison to more standard finite difference schemes by modeling a moving sinusoidal wave. It is not considered part of the convergence study. Example 1 is also the only example with a time dependent source term, and as such there is no steady state solution to converge to. So instead of a total variation diminishing scheme, a comparison is made between the time marching Runge-Kutta (2.20a) and centered difference schemes (1.26) after one second has elapsed. From [Table 1] it can be seen that even though neither scheme offers great performance, relatively speaking the WENO scheme is able to achieve a similar L_1-$norm$ as the centered difference scheme does with a CFL condition two hundred times smaller, and on average is an order of magnitude more accurate.



[FIGURE 12]

WENO approximation to (3.7a) at $t = 1$ and $N = 160$ (Figure 12 left) and centered difference approximation (Figure 12 right). The CFL condition is set to 4, which is quite large. The centered difference approximation is crumbling much more significantly than the WENO approximation.

Comparison of Various CFL Conditions on L_1-$norm$

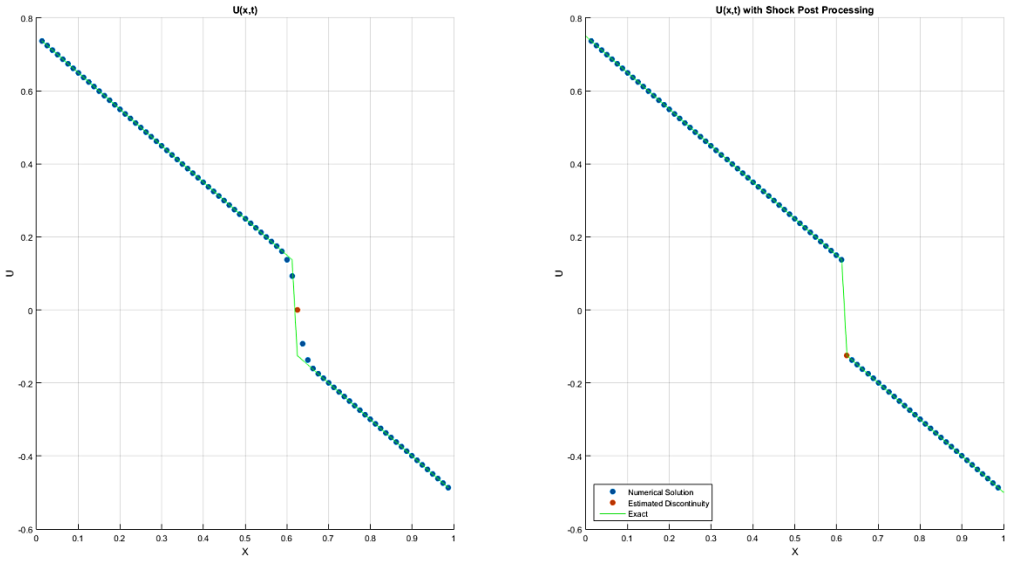
	Time Marching Runge-Kutta WENO				Centered Difference			
CFL	$N = 80$	$N = 160$	$N = 320$	$N = 640$	$N = 80$	$N = 160$	$N = 320$	$N = 640$
0.01	2.1423e-04	1.0932e-04	5.7976e-05	2.9373e-05	0.0404	0.0209	0.0106	0.0054
0.1	0.0022	0.0011	5.8444e-04	2.9416e-04	0.0505	0.0261	0.0133	0.0067
0.5	0.0112	0.0058	0.0029	0.0015	0.0947	0.0490	0.0249	0.0125
1	0.0222	0.0116	0.0059	0.0152	0.1473	0.0768	0.0392	0.0277
2	0.0461	0.0228	0.0165	-	0.2506	0.1306	0.0685	-

[TABLE 2]

A comparison of the L_1-$norm$ at different discretization and CFL settings for both the time-marching Runge-Kutta WENO and centered difference schemes. What the WENO scheme is able to achieve with a CFL condition of 2, and the centered difference scheme achieves with CFL of 0.01 for the same discretization.

Example 2:

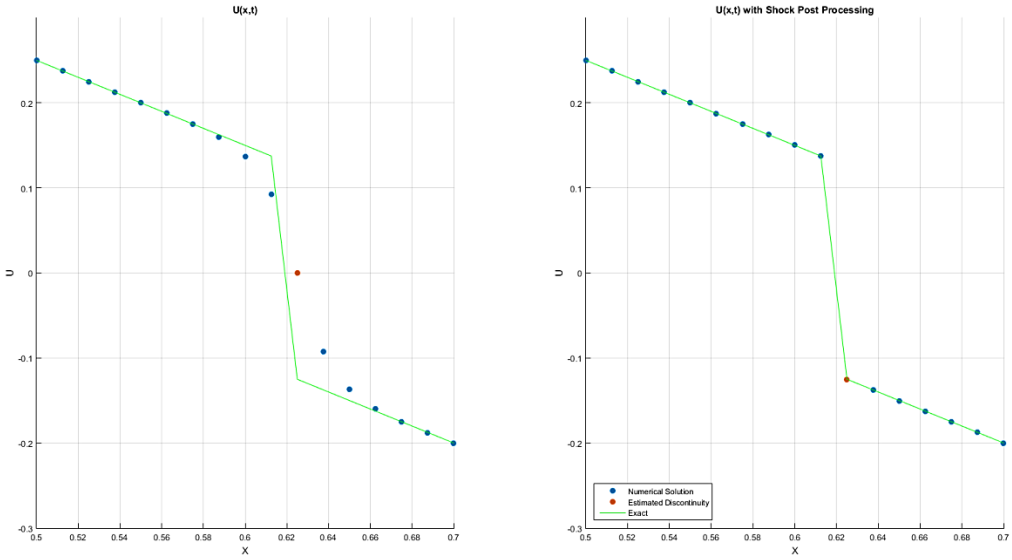
$$U_t - \left(\frac{1}{2}U^2\right)_x = U$$



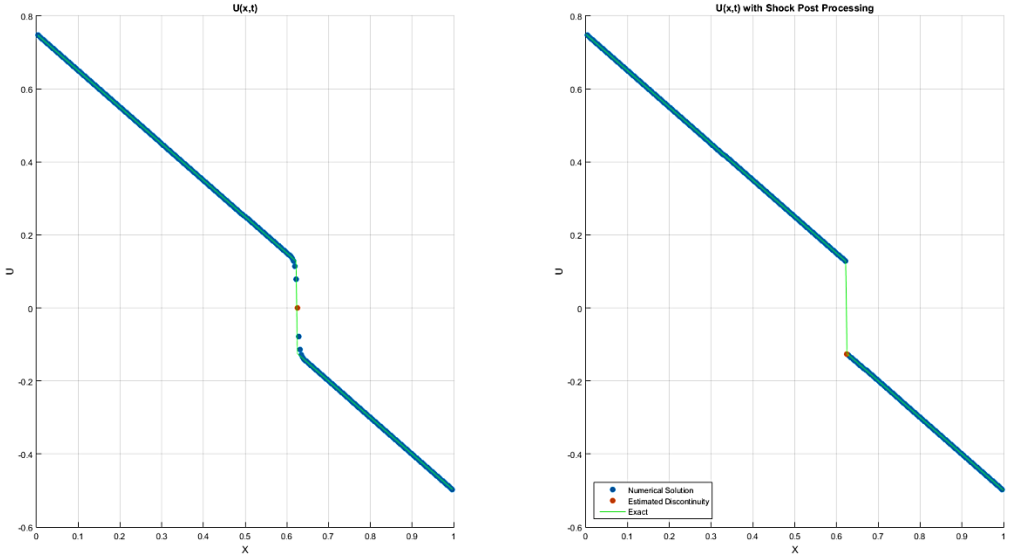
[FIGURE 13]

WENO approximation to (3.8a) with $N = 80$ pre- (Figure 13 left) and post-processing (Figure 13 right).

This is the inviscid Burger's Equation (1.7) with a source term (3.8a). This example has been seen before (Figure 4) and is now shown in a convergent steady state using WENO-3. Due to its linearity and the existence of a strong shock, an estimate to the discontinuity can be made and the discontinuous region can be refined to provide better resolution (Figure 11 right). Concerning the rates of convergence between the time-marching and fast-sweeping schemes, third-order accurate solutions are shown to have accelerated convergence potential using the fast-sweeping methodology. This equates to needing two to three times fewer iterations to reach the same degree of convergence, and with refinements to the extrapolation factor the overall algorithm can be made 80% more efficient than the time-marching variant.

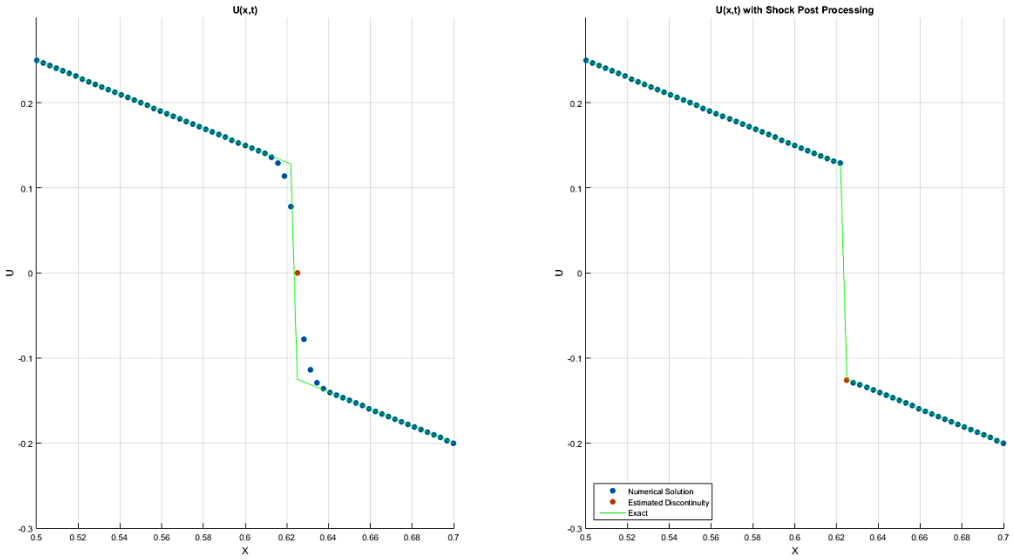


[FIGURE 14]

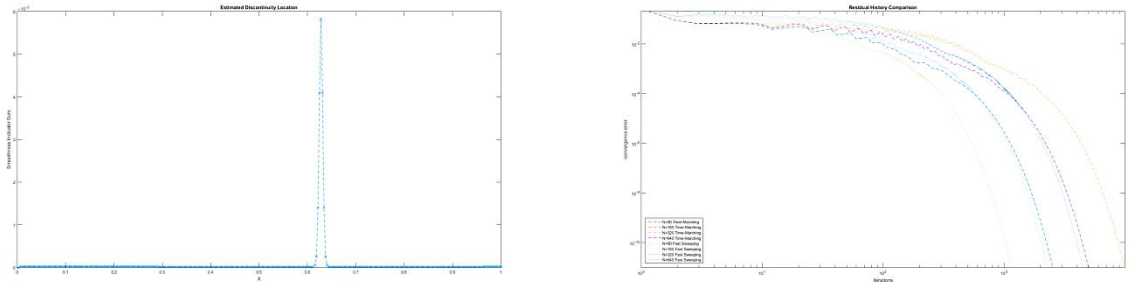


[FIGURE 15]

WENO approximation to (3.8a) with $N = 320$ pre- (Figure 14 and 15 left) and post-processing (Figure 14 and 15 right), with a close-up of the discontinuous region given in (Figure 15).



[FIGURE 16]



[FIGURE 17]

WENO approximation to (3.8a) with $N = 320$ pre- (Figure 16 left) and post-processing (Figure 16 right) in a close-up of the discontinuous region.. Estimated location of discontinuous points (Figure 17 left) and the residual history for time-marching and fast-sweeping schemes at $N = 80, 160, 320, 640$ (Figure 17 right) in a log-log plot.

Error and Iteration Count Comparison

N	L_∞ error	Order	L_1 error	Order	Iteration Count Time-Marching	Iteration Count Fast-Sweeping
40	4.6819e-04	-	3.1193e-05	-	1113	513
80	8.1285e-06	5.85	8.4507e-07	5.21	2098	964
160	1.3657e-07	5.89	5.6482e-08	3.90	3981	1831
320	1.2144e-08	3.49	4.2246e-09	3.74	7580	3478
640	4.7329e-10	4.68	1.5354e-10	4.78	14446	6590

[TABLE 3]

For all discretizations, a comparison is made of the error and order for the fast-sweeping and time-marching algorithms, including iteration count. The extrapolation factor is set to 1.

Extrapolation Factor Comparison on Iteration Count

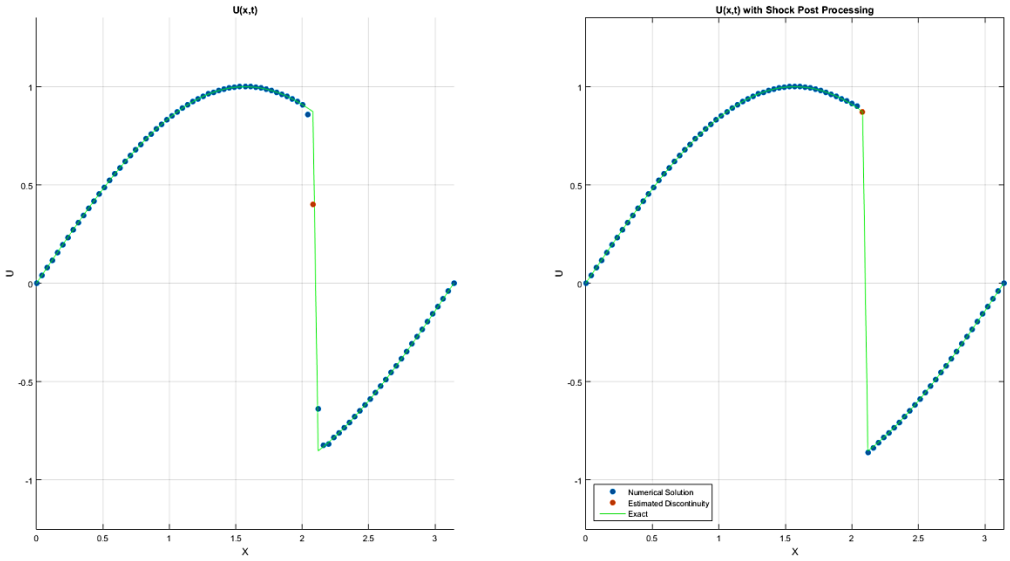
ω	$N = 80$	$N = 160$	$N = 320$	$N = 640$
0.9	1083	2055	3898	7392
1.0	964	1831	3478	6590
1.5	603	1147	2195	4187
2.0	422	802	1537	2954
2.1	397	753	1444	-
2.5	316	-	-	-
3.0	241	-	-	-
Optimal Iteration Reduction %	75%	58.87%	58.48%	55.17%

[TABLE 4]

For all discretizations, various extrapolation factors are used within the fast-sweeping Gauss-Seidel scheme in order to reduce iteration count. The percentage reduction is a comparison of the $\omega = 1$ iteration count and the minimum able to be achieved.

Example 3:

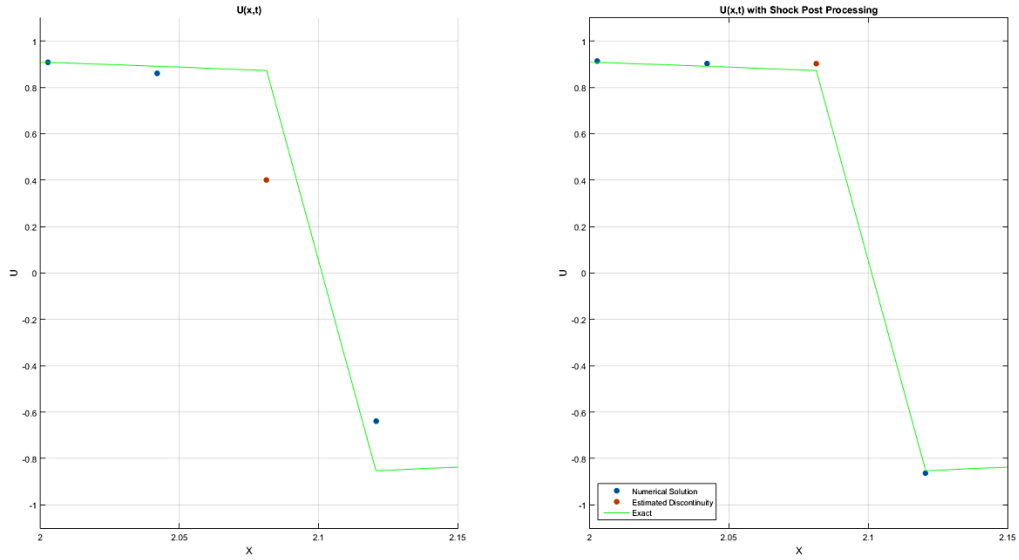
$$U_t + \left(\frac{1}{2}U^2\right)_x = sin(x)cos(x)$$



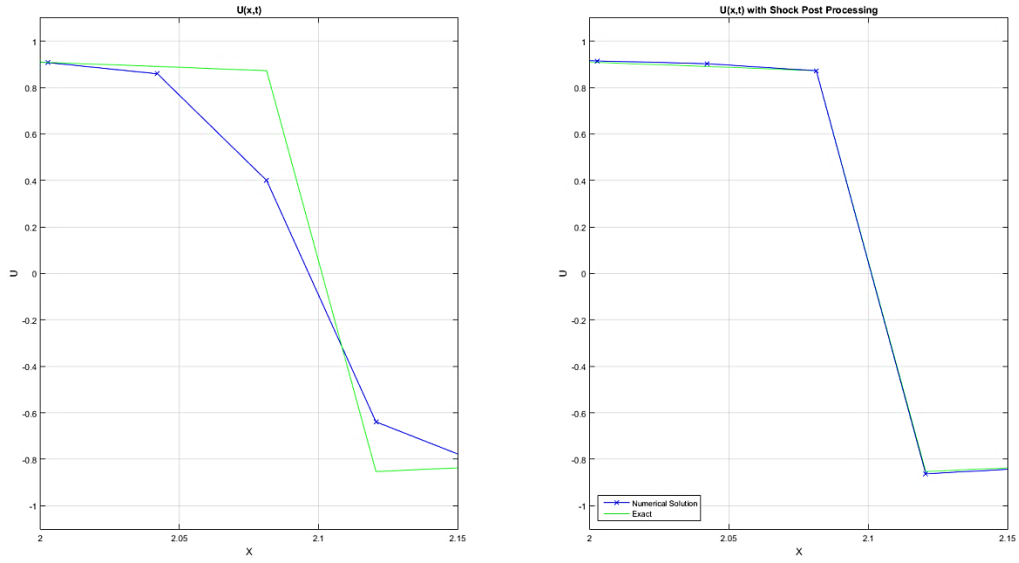
[FIGURE 18]

WENO approximation to (3.9a) with $N = 80$. Assuming a strong shock, the red point estimates its location, pre- (Figure 18 left) and post-processing (Figure 18 right).

The inviscid Burger's Equation with a source term (3.9a). Third-order accuracy is confirmed across multiple different mesh discretizations, and with the use of the most ambitious extrapolation factor overall efficiency can be increased by over 70%. A post-processing to the shock location is also applied to meshes of $N = 80$ and $N = 320$, and if a strong shock can be assumed, the overall accuracy of the function can be improved (Table 7). Scatter and line plots are given to showcase the potential of shock post-processing.

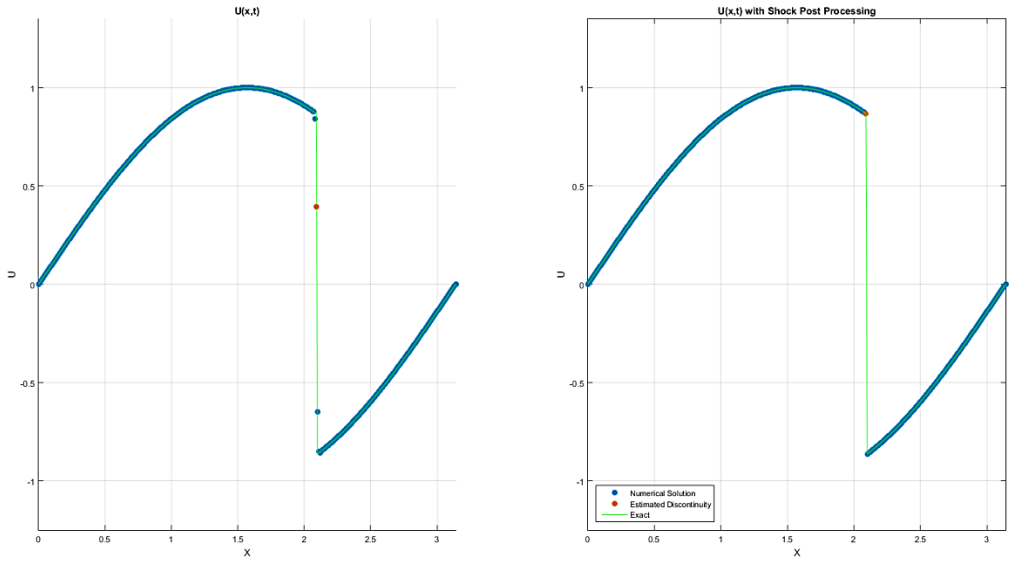


[FIGURE 19]

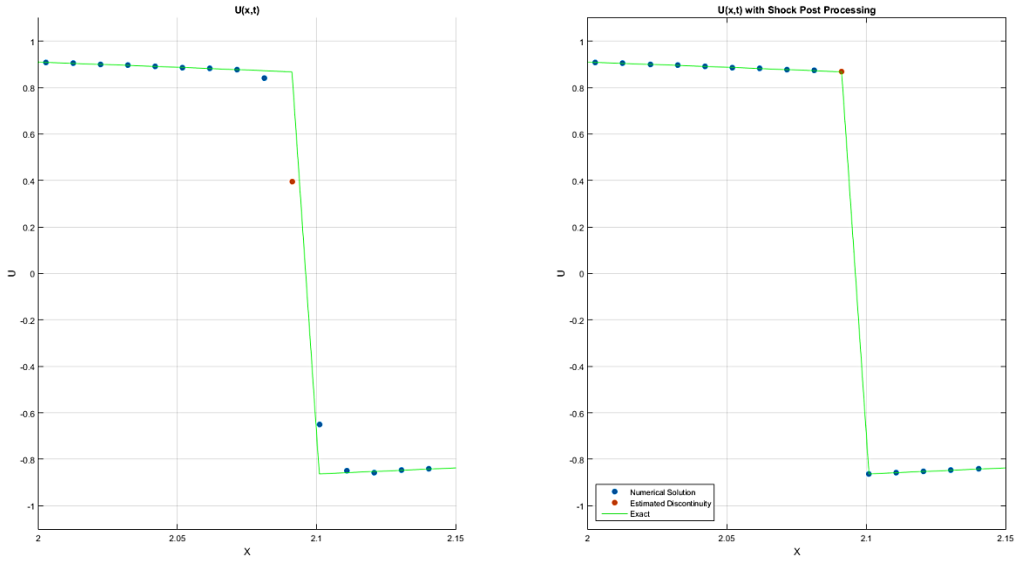


[FIGURE 20]

Close up of the WENO approximation with $N = 80$ pre- (Figure 19 and 20 left) and post-processing (Figure 29 and 20 right) around the shock location, connected by a line (Figure 20) to showcase the extent to which post processing helps resolve the shock location.

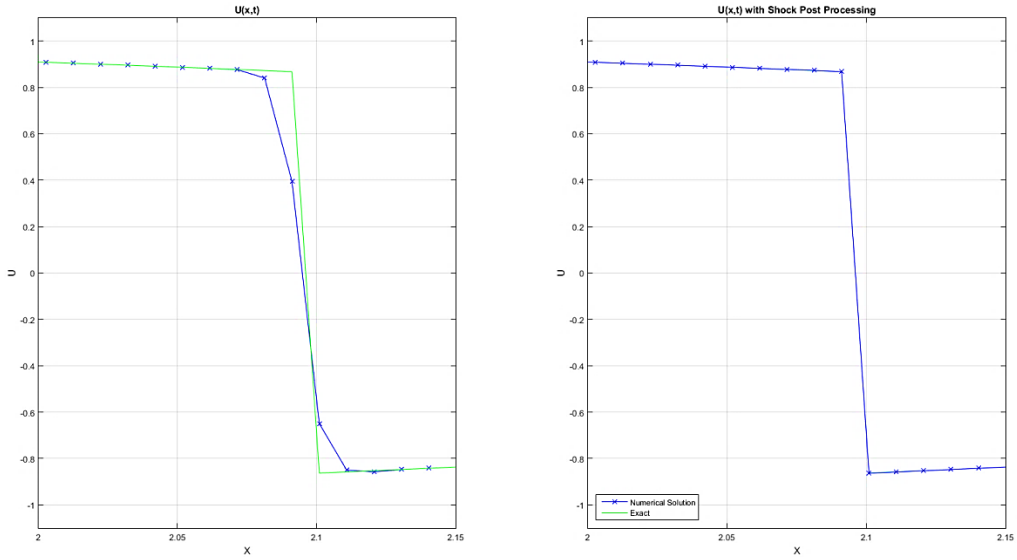


[FIGURE 21]

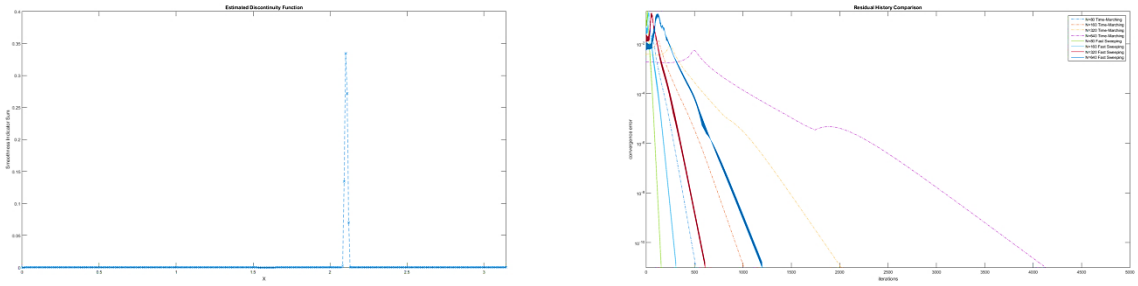


[FIGURE 22]

WENO approximation with $N = 320$ (Figure 21) and a close up of the same (Figure 22), with pre- and post-processing around the shock location.



[FIGURE 23]



[FIGURE 24]

Close up of the WENO approximation with $N = 320$ pre- and post-processing (Figure 23). Estimated location of discontinuous points (Figure 24 left) and the residual history for time-marching and fast-sweeping schemes at $N = 80, 160, 320, 640$ (Figure 24 right) in a log plot.

Error and Iteration Count Comparison

N	L_∞ error	Order	L_1 error	Order	Iteration Count Time-Marching	Iteration Count Fast-Sweeping
80	0.0048	-	1.6596e-04	-	482	244
160	2.0466e-04	4.55	5.7707e-06	4.84	941	484
320	2.5783e-06	6.31	3.9717e-07	3.86	1839	952
640	1.7871e-07	3.85	4.0009e-08	3.31	3592	1951

[TABLE 5]

For all discretizations, a comparison is made of the error and order for the time-marching and fast-sweeping algorithms, along with iteration counts. The extrapolation factor is set to 1.

Extrapolation Factor Comparison on Iteration Count

ω	$N = 80$	$N = 160$	$N = 320$	$N = 640$
0.9	274	542	1063	2201
1.0	244	484	952	1951
1.25	192	380	746	1495
1.5	154	308	608	1197
1.75	-	256	510	1010
2.0	-	-	-	-
Optimal Iteration Reduction %	36.88%	47.12%	46.43%	48.23%

[TABLE 6]

For all discretizations, various extrapolation factors are used within the fast-sweeping Gauss-Seidel scheme in order to reduce iteration count. The percentage reduction is a comparison of the $\omega = 1$ iteration count and the minimum able to be achieved.

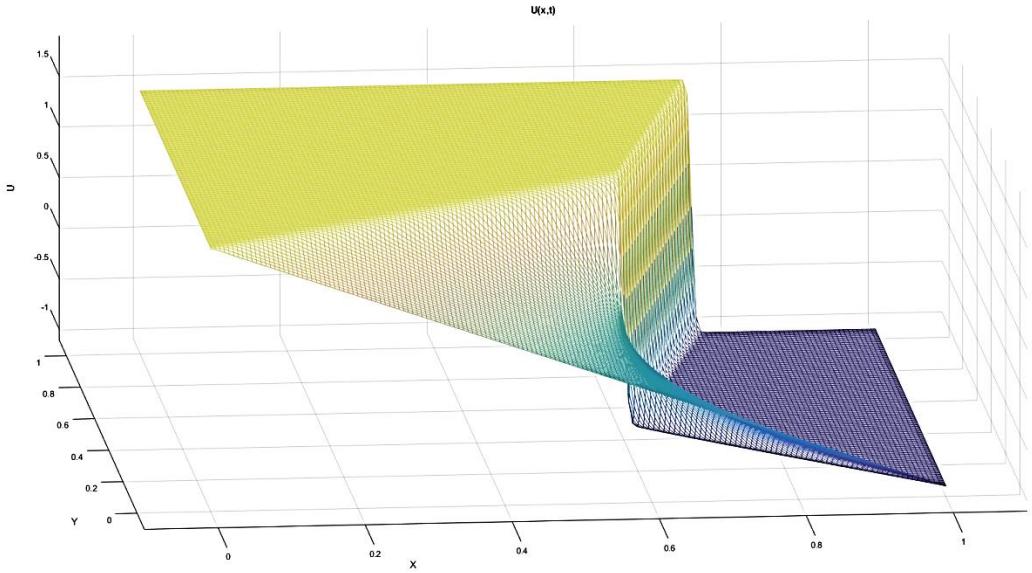
N	L_1 error	L_1 error Post Processing
80	0.0032	5.2207e-04
160	0.0016	6.4930e-05
320	8.0679e-04	9.0798e-06
640	4.0084e-04	1.9314e-06

[TABLE 7]

A comparison of the L_1-$norm$ at different discretizations for the shock pre- and post-processing procedures. The L_1-$norm$ is taken across the entire domain, excluding only the singular estimated discontinuous point.

Example 4:

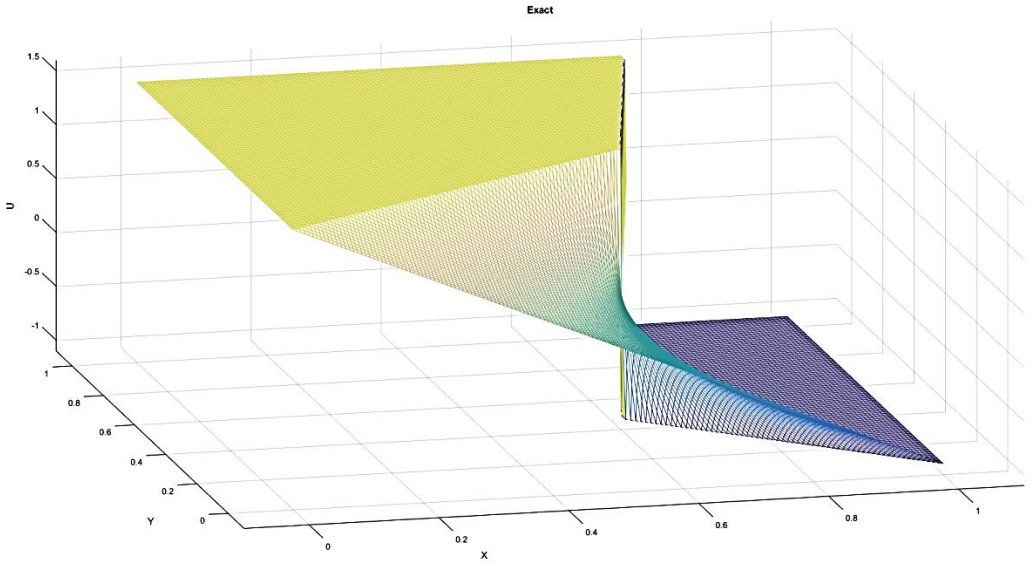
$$U_t + \left(\frac{1}{2}U^2\right)_x + U_y = 0$$



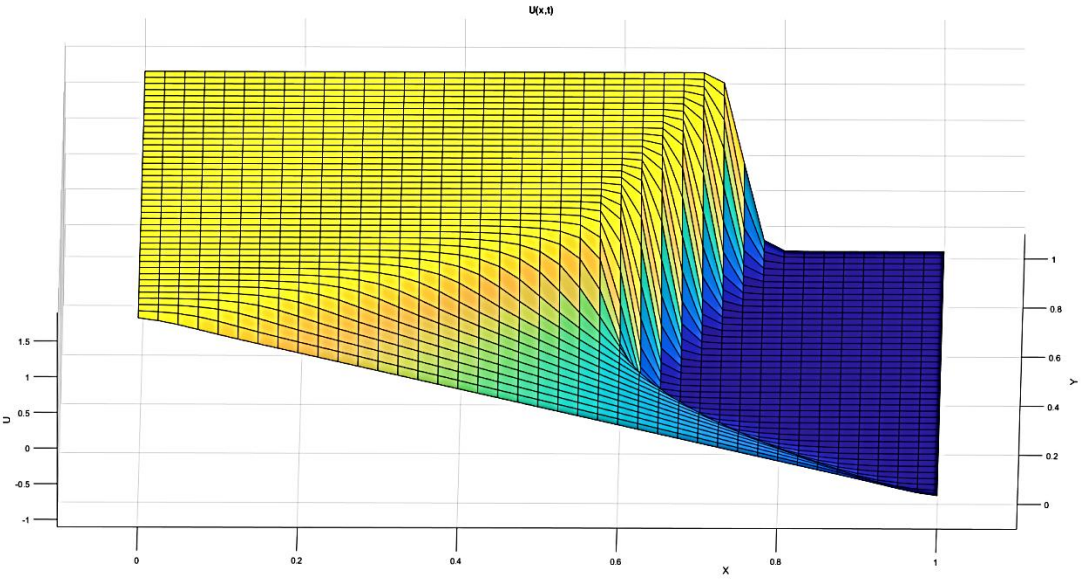
[FIGURE 25]

WENO approximation to (3.10a) with $N = 80$ and $M = 80$.

The first of three 2-dimensional examples (3.10a). The solution exhibits areas of smooth and discontinuous transitions, and because a strong shock cannot be assumed across the entire discontinuous region, a range of discontinuous locations (Figure 28) is estimated along with estimates for the singular point (Figure 29). These are shown in the context of the contour lines (Figure 30) and the characteristic lines (Figure 31), both of which refer to similar information regarding the spatial dynamics of the steady state solution. Various cross-sections are taken in both the x- and y- directions, and the fast-sweeping approach with a maximized extrapolation factor is around 50% more efficient than the time-marching scheme. The two schemes are nearly identical for ω near unity.

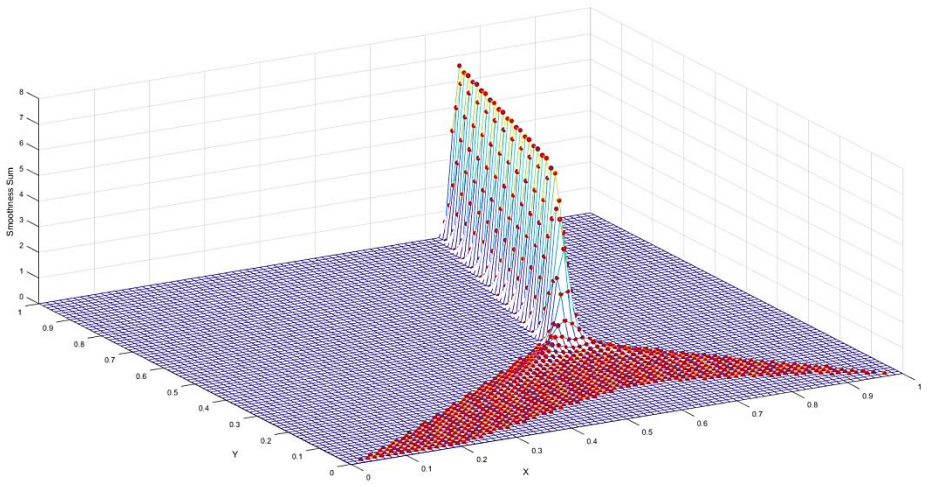


[FIGURE 26]

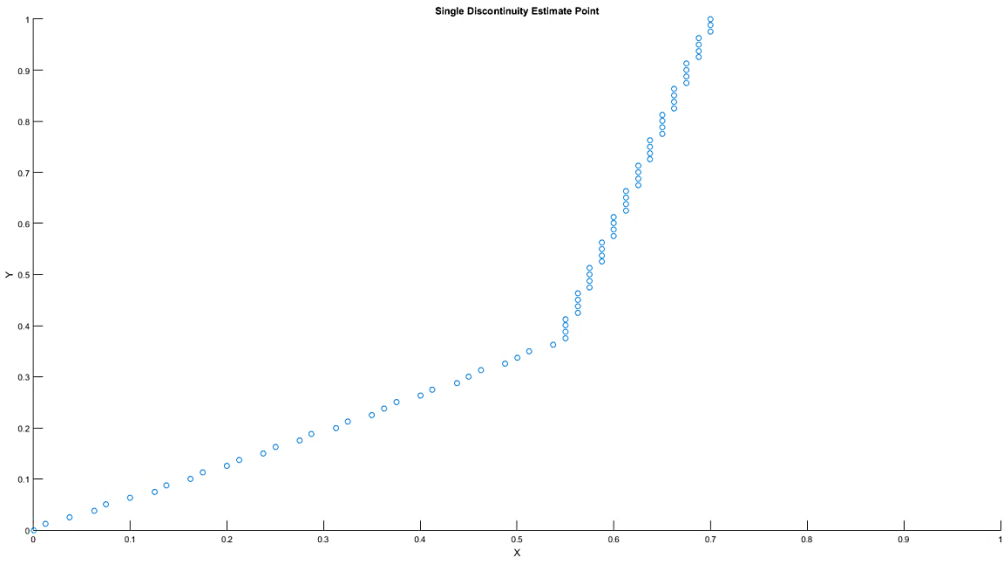


[FIGURE 27]

The exact solution (Figure 26) at a tighter angle and a WENO approximation (Figure 27) with $N = 80$ and $M = 80$ using a surface plot.

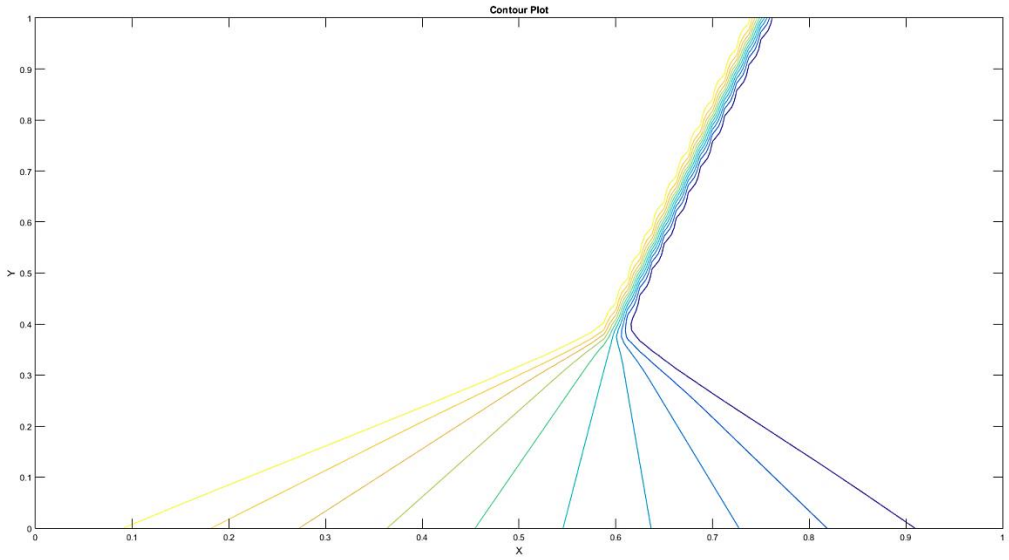


[FIGURE 28]

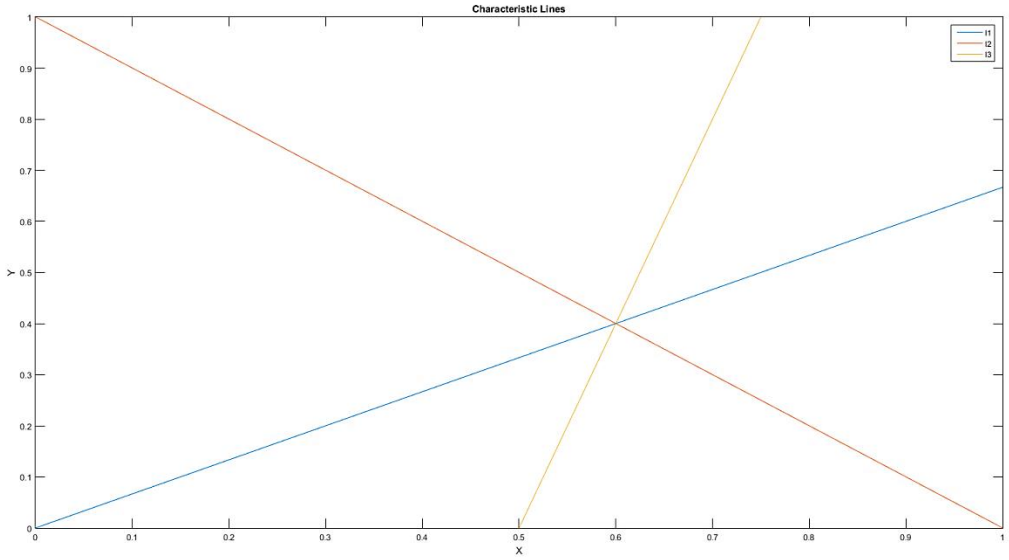


[FIGURE 29]

All candidate point values for estimated discontinuities (Figure 28) and a plot of the estimated location of discontinuity (Figure 29) assuming a single point.

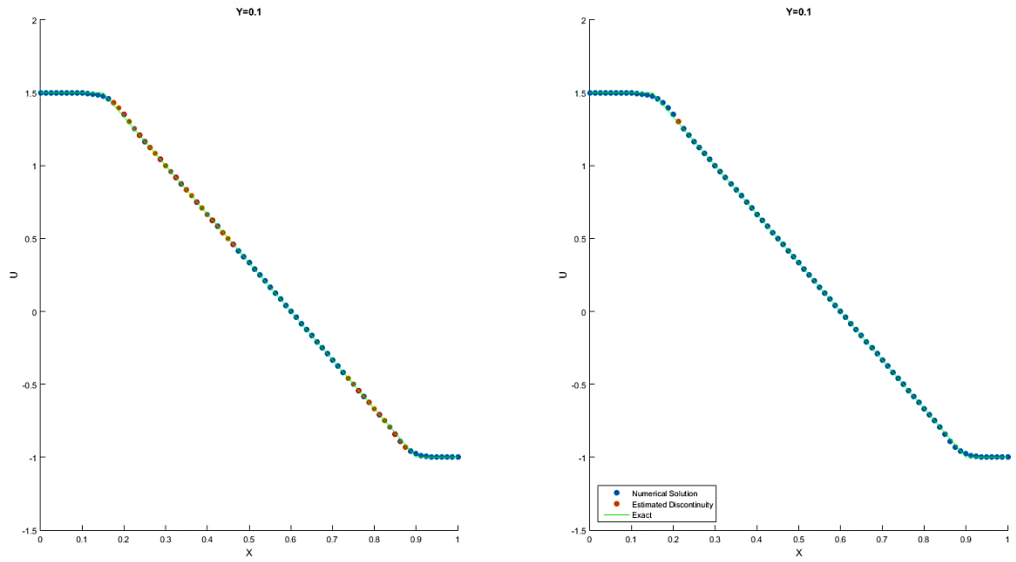


[FIGURE 30]

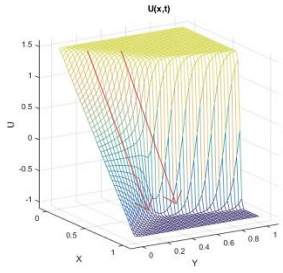


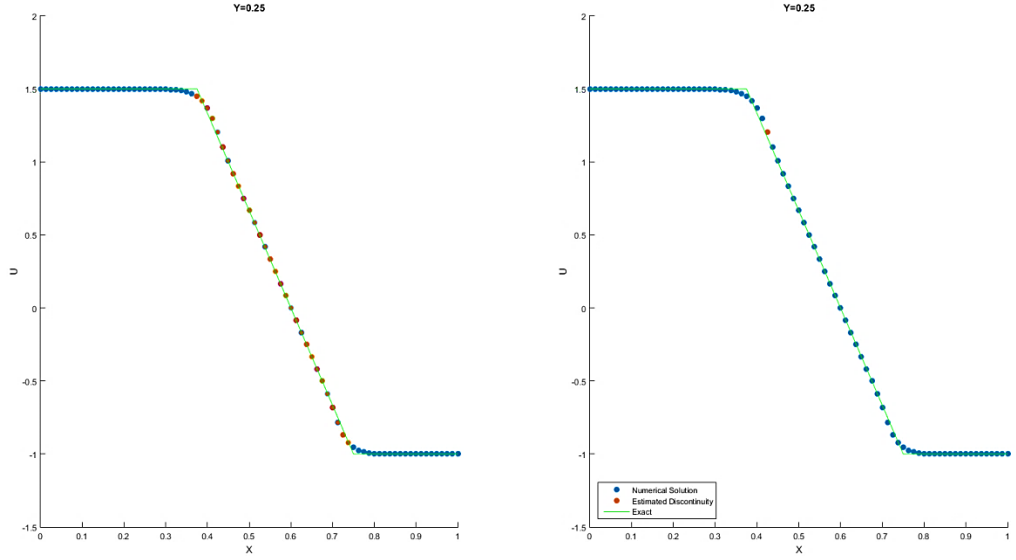
[FIGURE 31]

Contour lines (Figure 30) and the characteristic lines (Figure 31) which the solution adheres to. Note how the three previous figures (Figure 29, 30, and 31) all represent similar information.

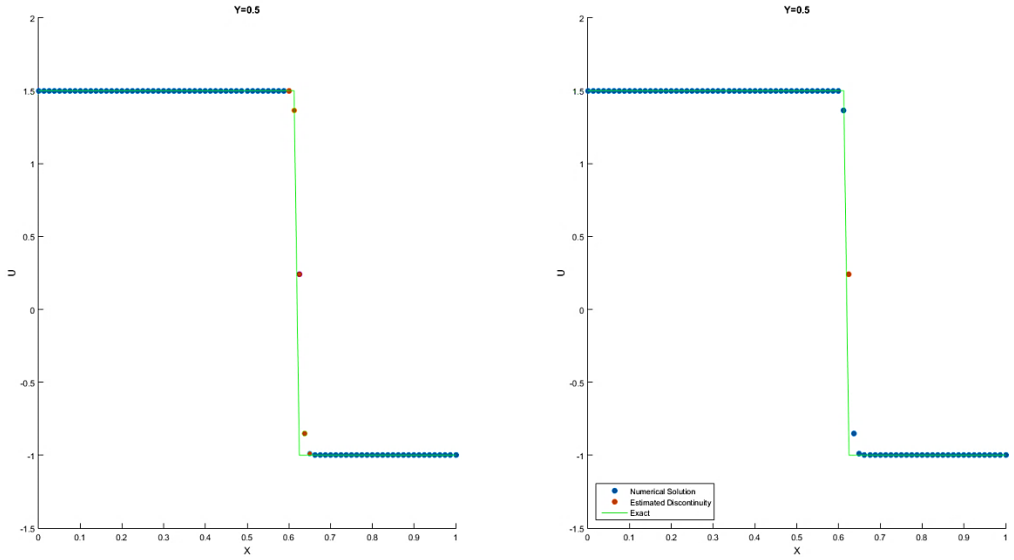


[FIGURE 32]

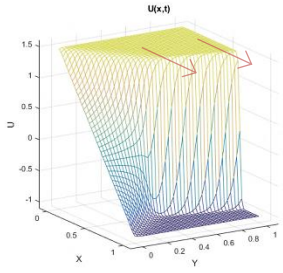


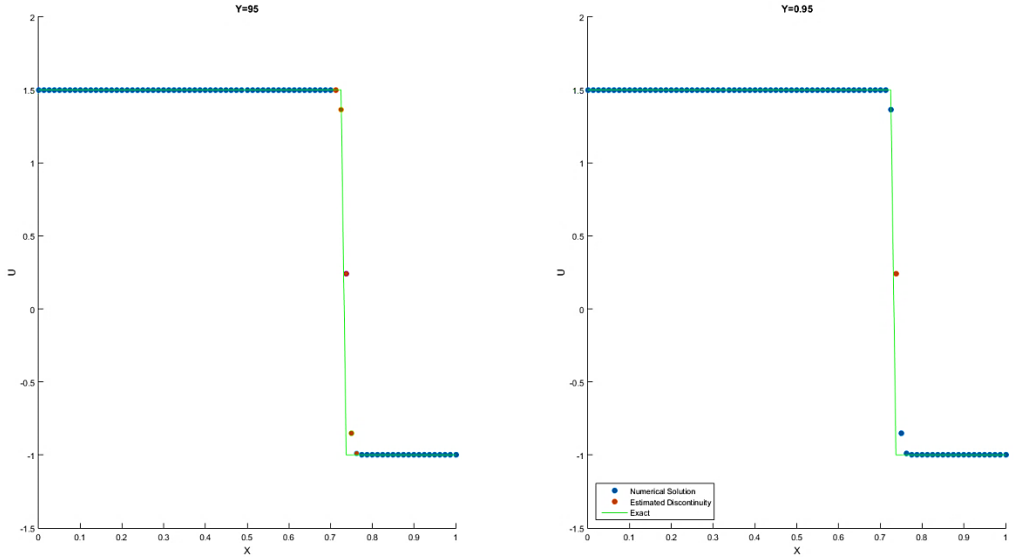


[FIGURE 33]

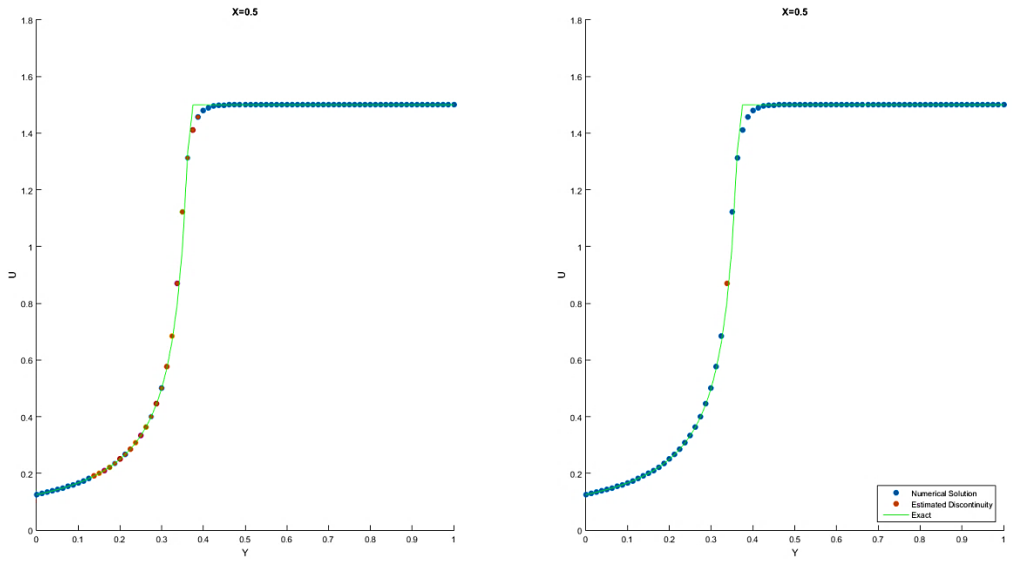


[FIGURE 34]

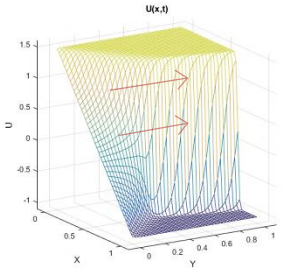


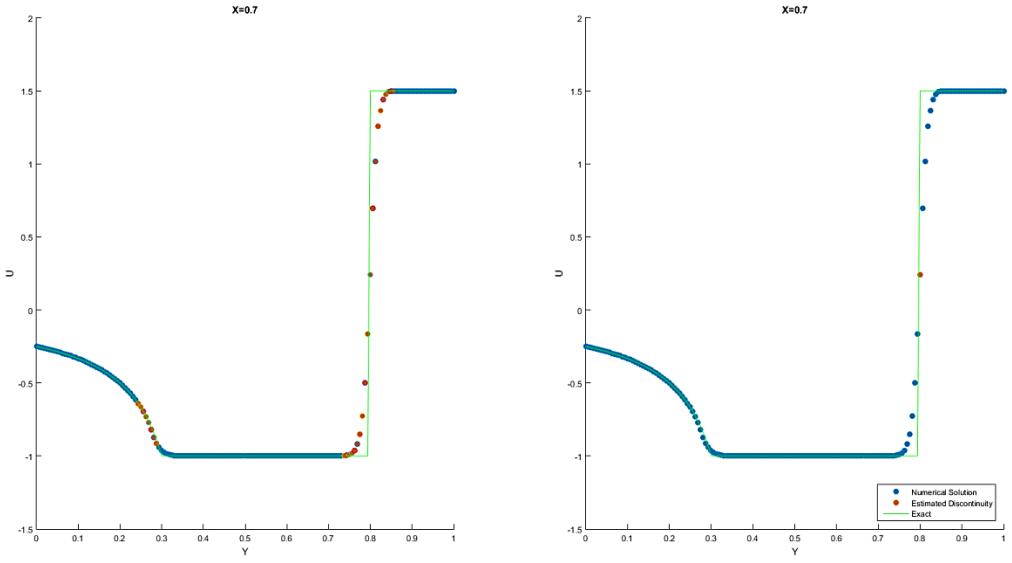


[FIGURE 35]



[FIGURE 36]





[FIGURE 37]

Various cross-sections at $Y = 0.1, 0.25, 0.5, 0.75$ (Figure 32, 33, 34, 35) and $X = 0.5, 0.7$ (Figure, 36 and 37). A spread is given for the estimated location of discontinuity (left image), and a single point is given in the event of a strong shock (right image) with $N = 80$ and $M = 80$.

Error and Iteration Count Comparison

N	L_∞ error	Order	L_1 error	Order	Iteration Count Time-Marching	Iteration Count Fast-Sweeping
20	0.0043	-	2.4000e-04	-	152	156
40	3.7521e-05	6.84	2.1755e-06	6.78	242	236
80	1.2934e-06	4.86	2.4847e-08	6.45	418	410
160	2.2077e-09	9.19	2.3475e-11	10.05	767	737
320	9.2149e-15	17.87	2.2152e-15	13.37	1436	1391

[TABLE 8]

For all discretizations, a comparison is made of the error and order for the fast-sweeping and time-marching algorithms, including iteration count. The extrapolation factor is set to 1.

Extrapolation Factor Comparison on Iteration Count

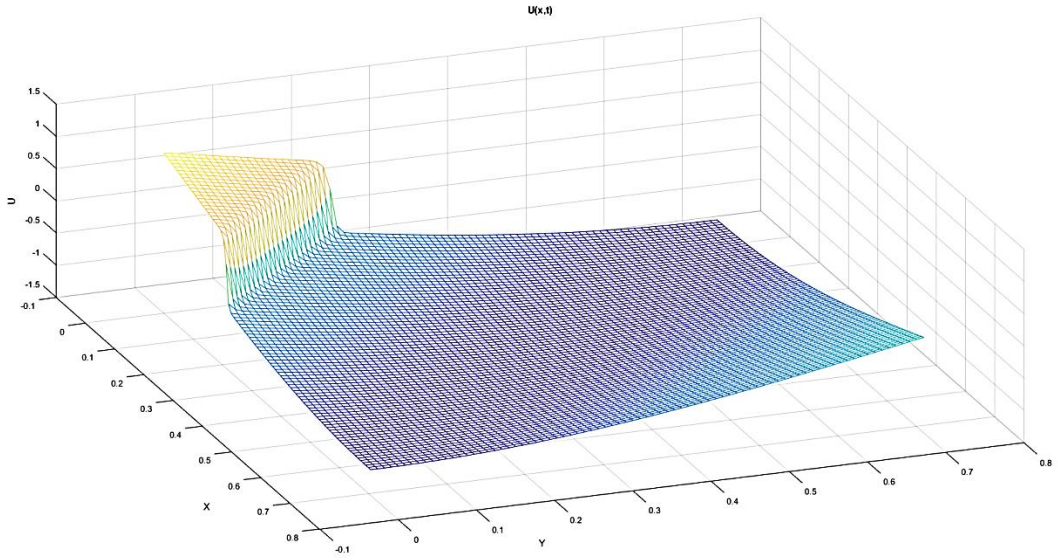
ω	$N = 20$	$N = 40$	$N = 80$	$N = 160$
0.9	168	265	450	828
1.0	156	236	410	737
1.5	110	158	268	482
2.0	88	126	202	358
2.25	82	114	179	318
2.5	-	-	-	-
Optimal Iteration Reduction %	47.44%	51.69%	56.34%	56.85%

[TABLE 9]

For all discretizations, various extrapolation factors are used within the fast-sweeping Gauss-Seidel scheme in order to reduce iteration count. The percentage reduction is a comparison of the $\omega = 1$ iteration count and the minimum able to be achieved.

Example 5:

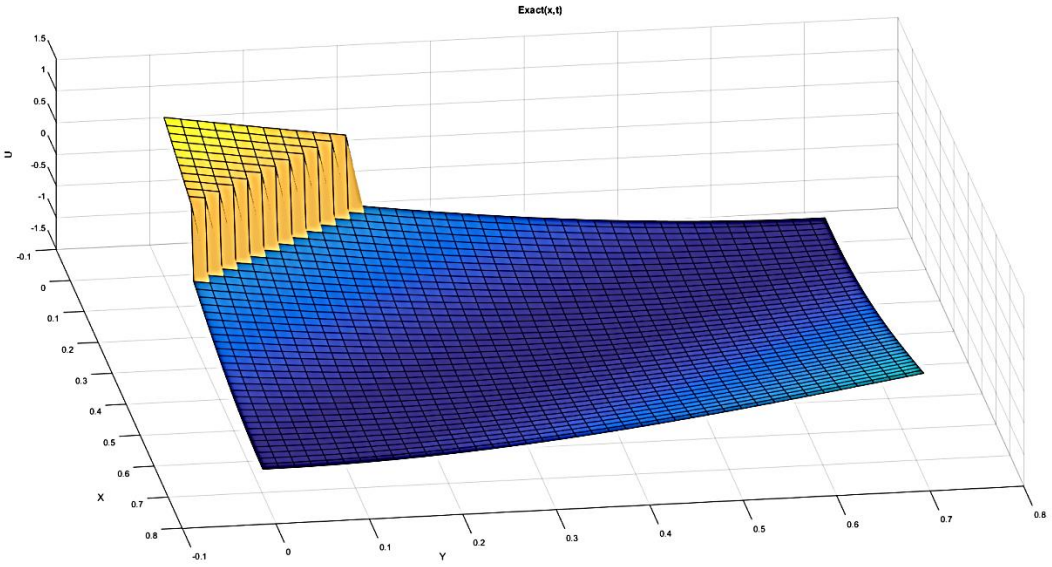
$$U_t + \left(\frac{U^2}{2\sqrt{2}}\right)_x + \left(\frac{U^2}{2\sqrt{2}}\right)_y = -U\pi cos\left(\pi\frac{x+y}{\sqrt{2}}\right)$$



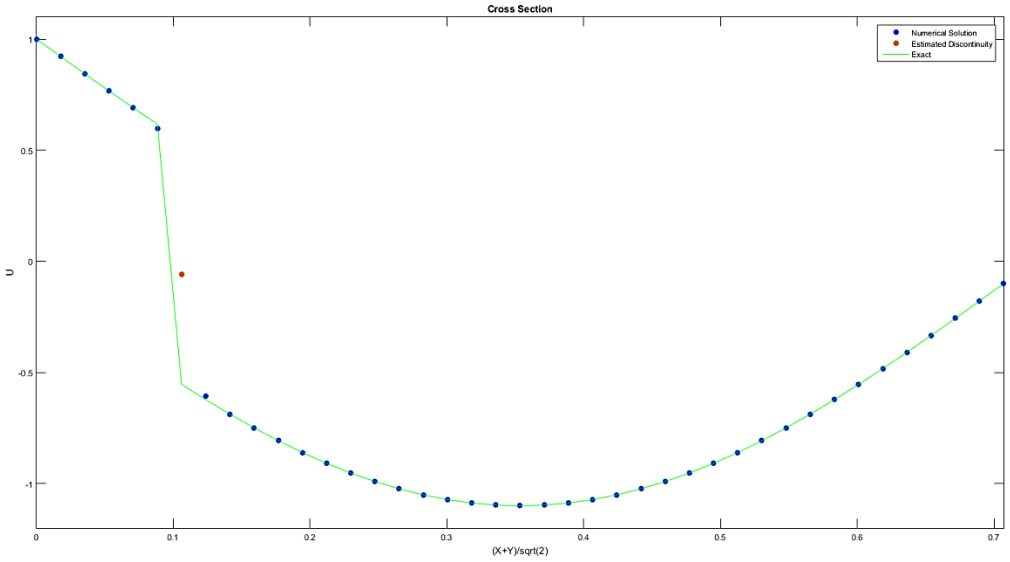
[FIGURE 38]

WENO approximation to (3.11a) with $N = 80$ and $M = 80$.

The second 2-dimensional example is another deformed sheet (3.11a) with a rich smooth region and an area of strong discontinuity. The line plots without any discontinuity estimate (Figure 40 and 41) are approximations at $N = 40$ and $N = 80$ along the long diagonal from left-to-right. Estimates to the discontinuous points are shown (Figure 43) in relation to the contour lines (Figure 44) to show their similar nature, as they are plots of the same information essentially. The discontinuity is then traversed at three different locations in the y-plane, and overall efficiency of the fast-sweeping scheme is shown to be upward of 70% over that of the time-marching method. This turned out to be the most demanding example to solve at higher mesh discretizations (note the iteration count of twenty-thousand out of nowhere) resulting in lengthy runtimes for the Matlab code. This is because slight oscillations in the solution are present for quite some time, and especially in the $N = 320$, where nearly after an entire day of calculations did the solution converge.

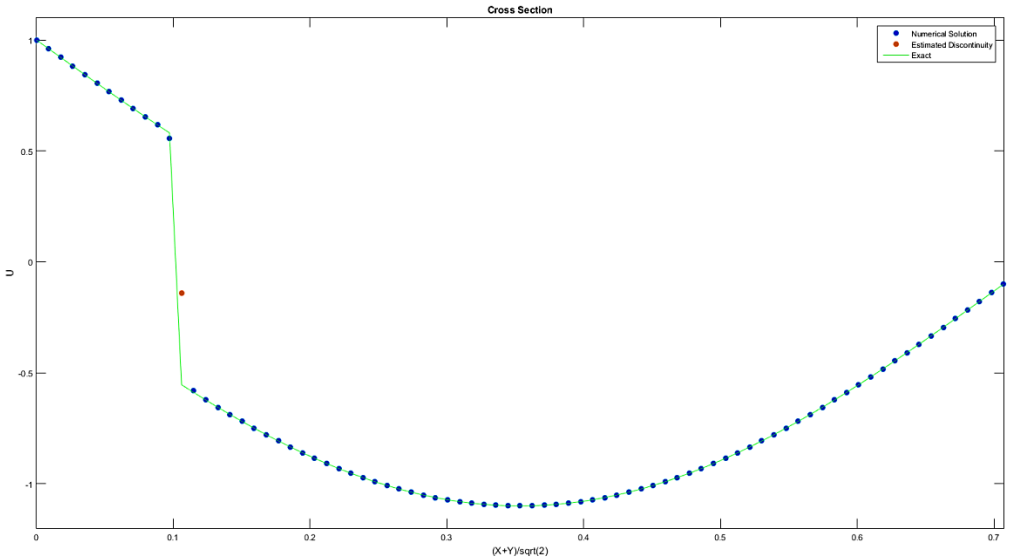


[FIGURE 39]

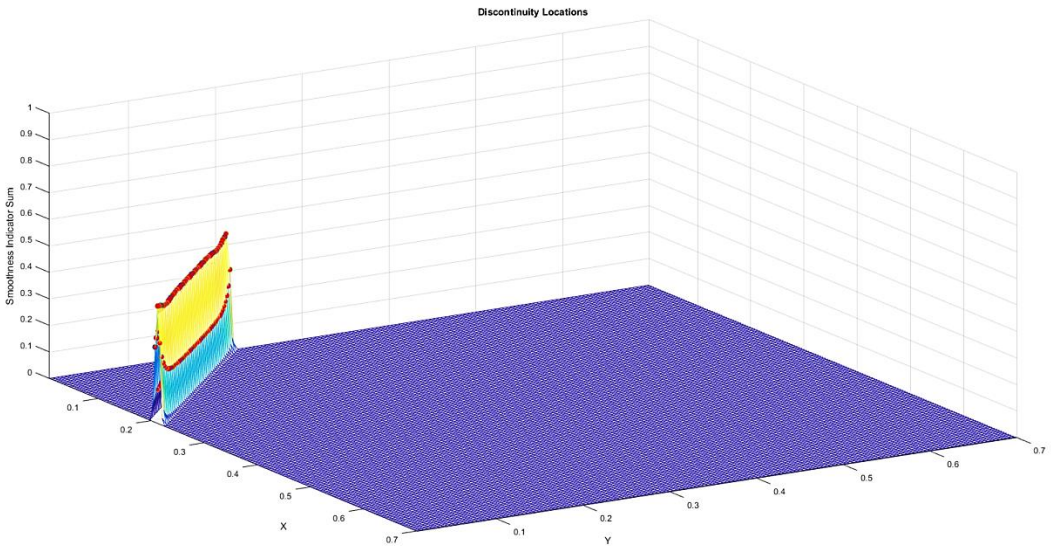


[FIGURE 40]

The exact solution (Figure 39) and a left-to-right plot of the diagonal (Figure 40) with $N = 40$ and $M = 40$.

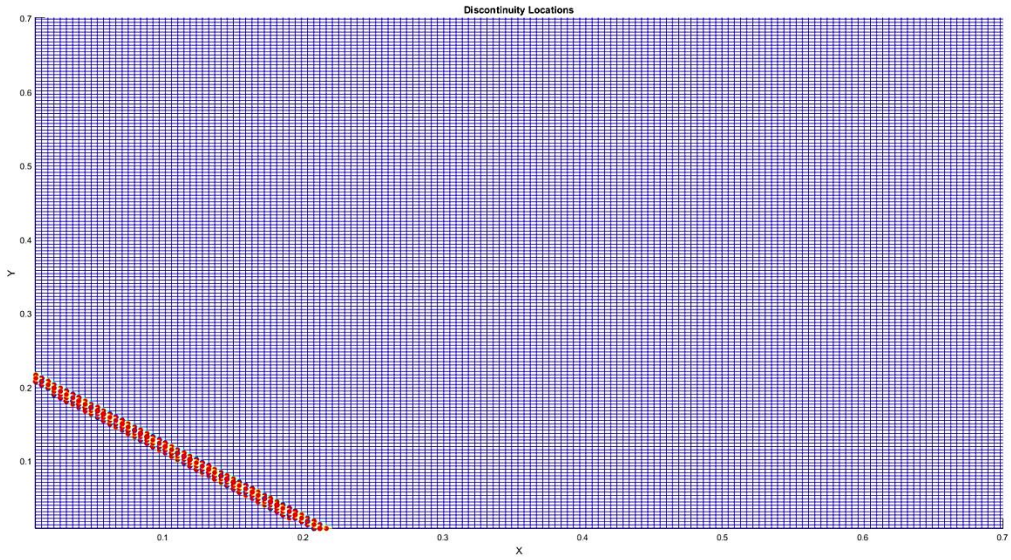


[FIGURE 41]

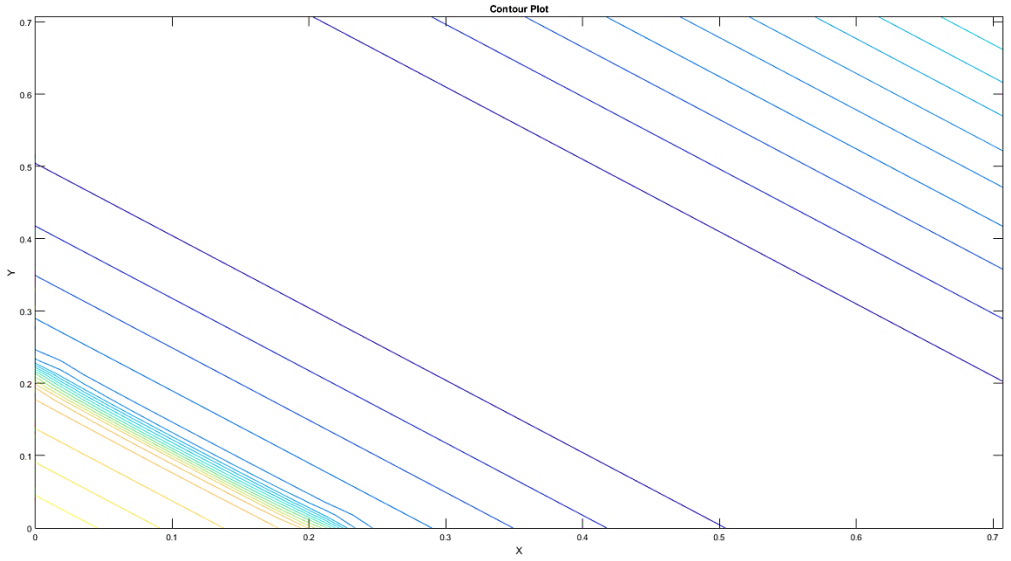


[FIGURE 42]

A left-to-right plot of the diagonal (Figure 41) with $N = 80$ and $M = 80$, and a mesh estimation for possible discontinuous locations (Figure 42) with $N = 160$ and $M = 160$.

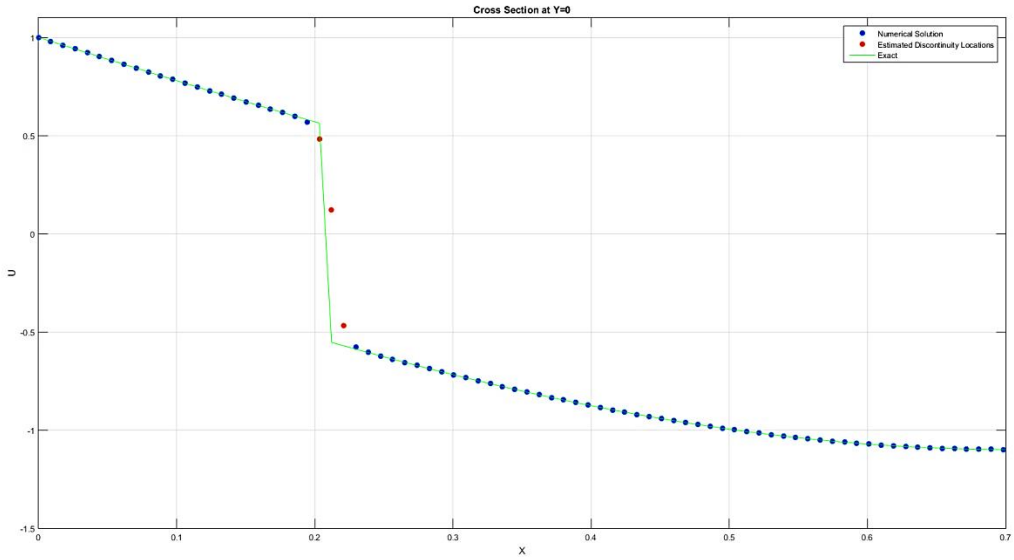


[FIGURE 43]

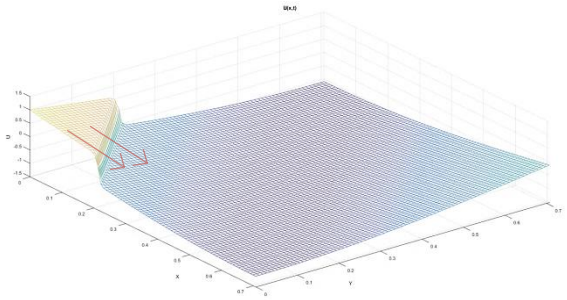


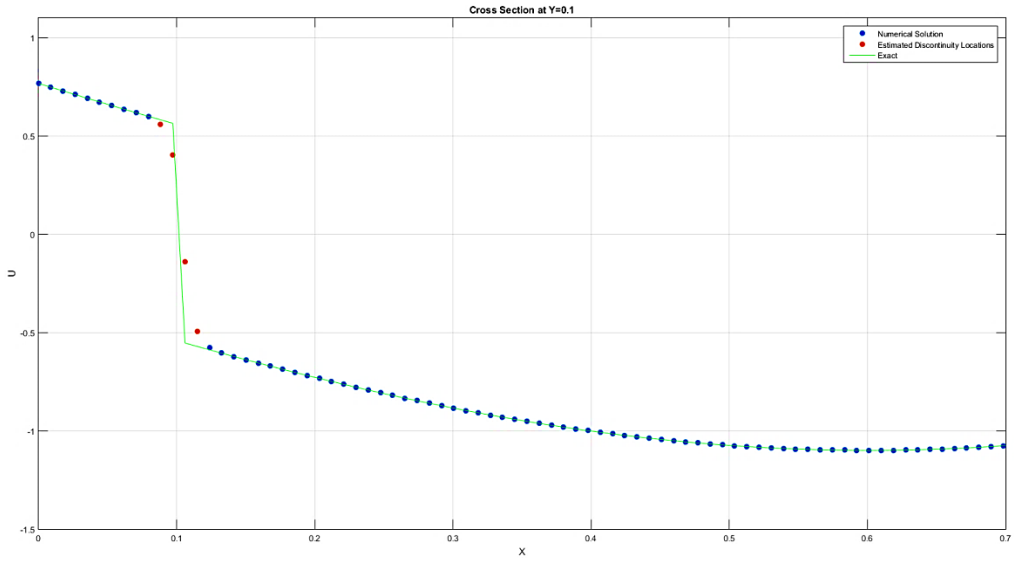
[FIGURE 44]

A top-down perspective of the candidate discontinuity point values (Figure 43), and the contour lines of the solution (Figure 44).

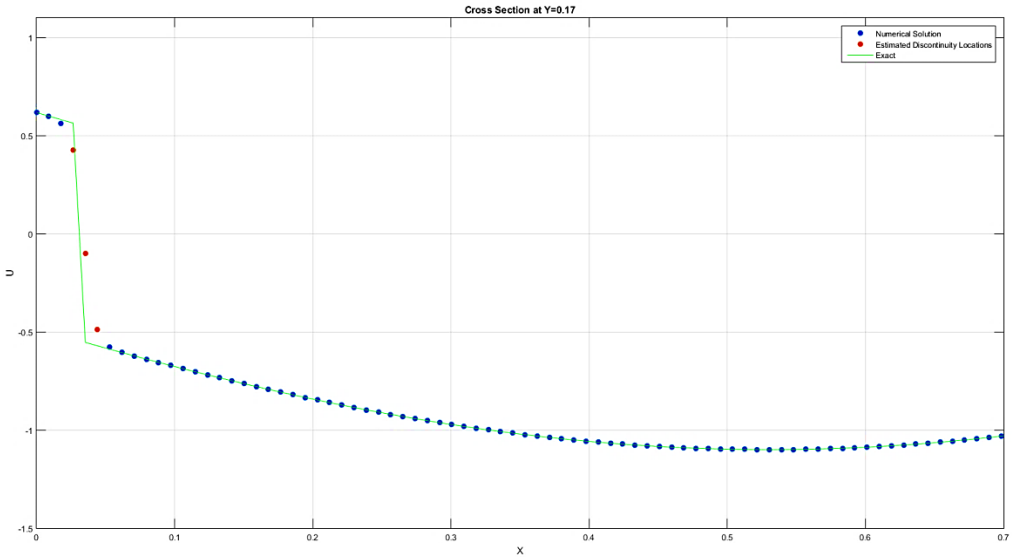


[FIGURE 45]

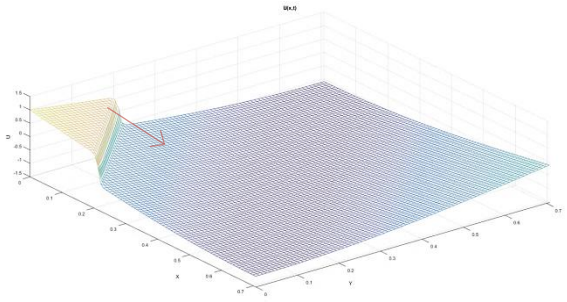




[FIGURE 46]



[FIGURE 47]



Various cross-sections at $Y = 0, 0.1, 0.17$ (Figure 45, 46, and 47). A spread is given for the estimated location of discontinuity.

Error and Iteration Count Comparison

N	L_∞ error	Order	L_1 error	Order	Iteration Count Time-Marching	Iteration Count Fast-Sweeping
40	3.2931e-04	-	4.0000e-05	-	975	726
80	2.9855e-05	3.4634	4.7446e-06	3.07	1828	1341
160	1.9848e-06	3.9109	3.8745e-07	3.61	Not Convergent	2725
320	1.6148e-07	3.6196	4.5116e-08	3.10	Not Convergent	20371

[TABLE 10]

For all discretizations, a comparison is made of the error and order for the fast-sweeping and time-marching algorithms, including iteration count. The extrapolation factor is set to 1.

Extrapolation Factor Comparison on Iteration Count

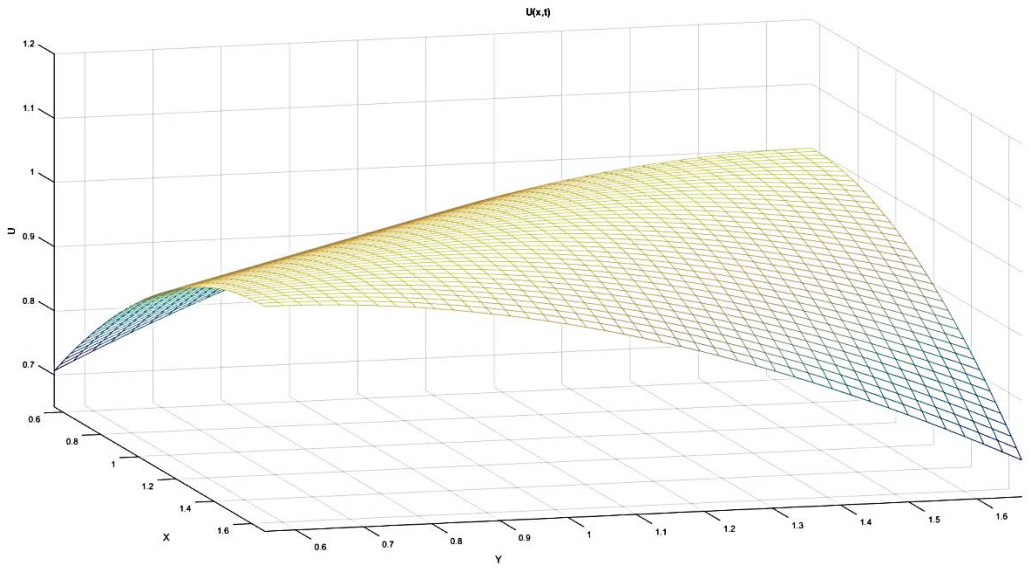
ω	$N = 20$	$N = 40$	$N = 80$
0.9	543	977	1826
1.0	481	865	1615
1.5	290	523	959
2	192	345	611
2.2	164	293	513
2.75	109	-	-
Optimal Iteration Reduction %	77.34%	66.13%	68.23%

[TABLE 11]

For all discretizations, various extrapolation factors are used within the fast-sweeping Gauss-Seidel scheme in order to reduce iteration count. The percentage reduction is a comparison of the $\omega = 1$ iteration count and the minimum able to be achieved.

Example 6:

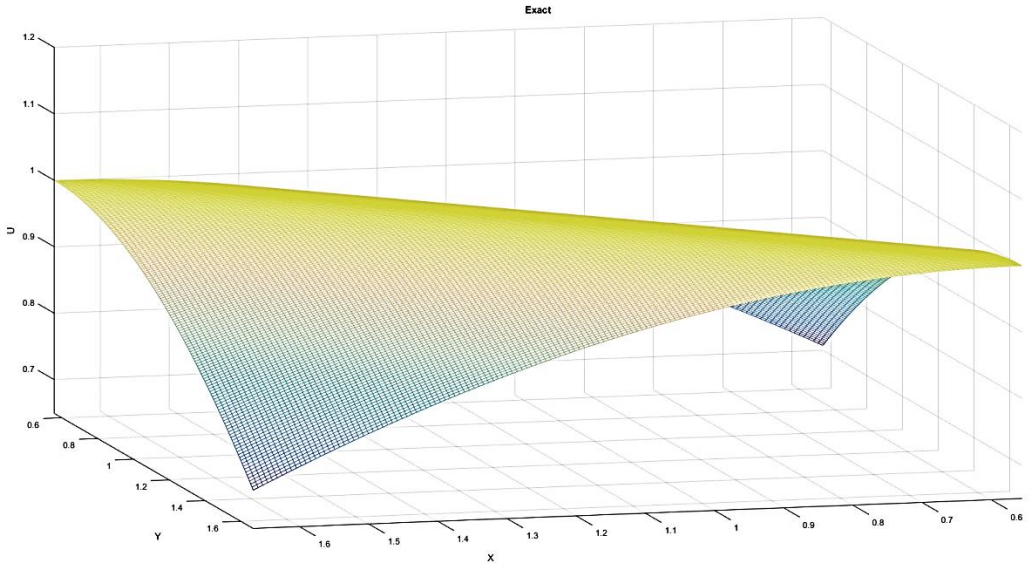
$$U_t + \left(\frac{U^2}{2\sqrt{2}}\right)_x + \left(\frac{U^2}{2\sqrt{2}}\right)_y = sin\left(\frac{x+y}{\sqrt{2}}\right) cos\left(\frac{x+y}{\sqrt{2}}\right)$$



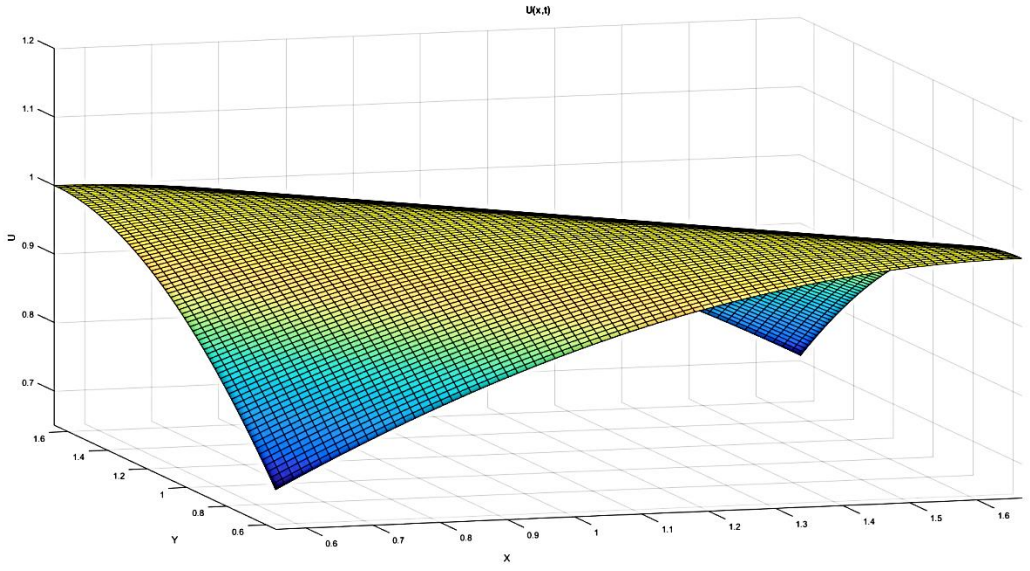
[FIGURE 48]

WENO approximation to (3.12a) with $N = 40$ and $M = 40$.

The final 2-dimensional example (3.12a), which is now a smooth sheet without any discontinuities. A similar investigation reveals that even under these more simple circumstances, efficiency improvements were still seen to be in the 70% range. Two cross-sections are taken to compare accuracy (Figure 53 and 54), with another cross diagonal plot taken as well (Figure 52). To return to the earlier discussions of CFL conditions, a value of 1 was used as well, to show yet another variable that can be tweaked to boost iterative convergence. As in [Figure 4], too large a CFL condition however results in divergence (Figure 55 and 56].

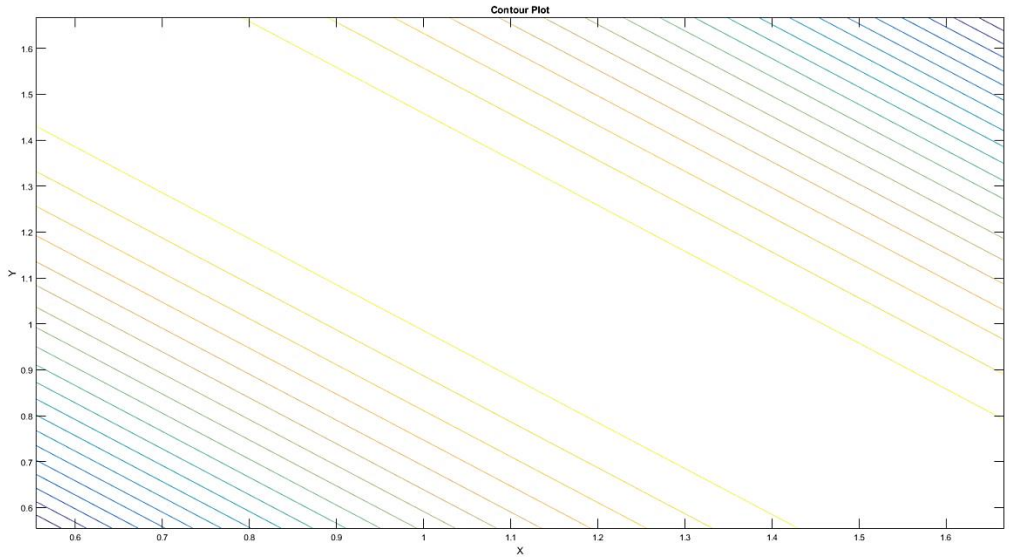


[FIGURE 49]

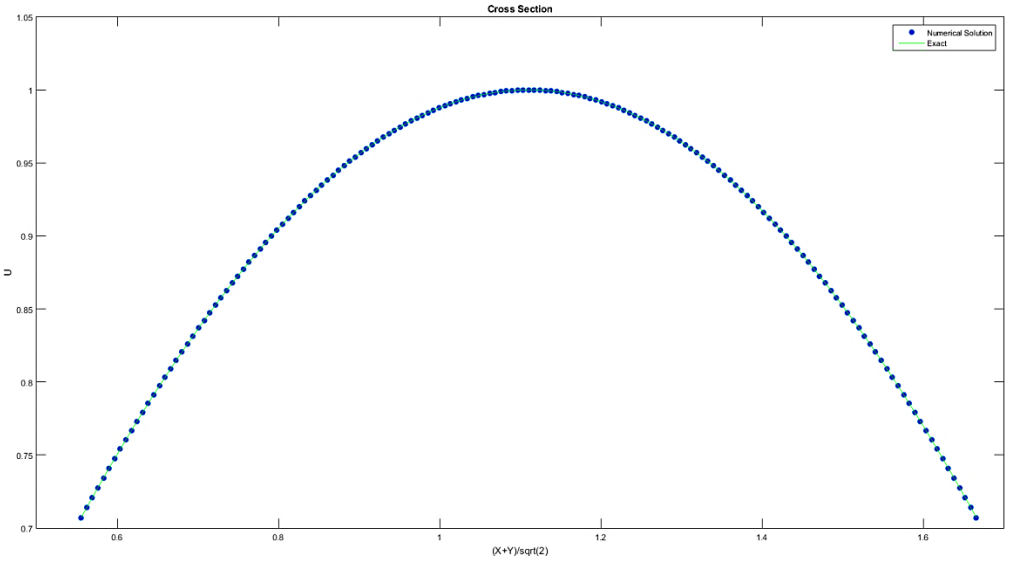


[FIGURE 50]

The exact solution (Figure 49) and an alternate angle to the WENO approximation (Figure 50) with $N = 80$ and $M = 80$.

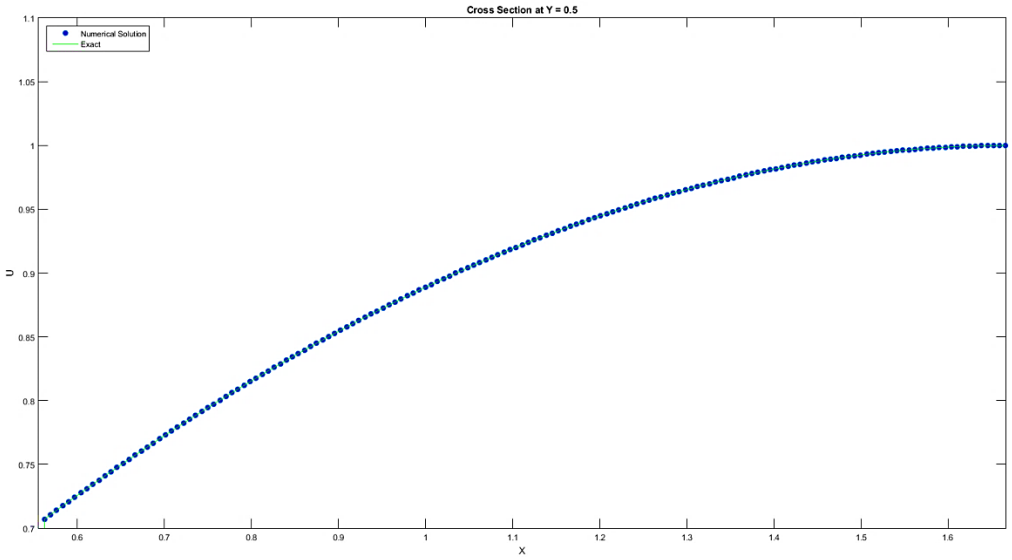


[FIGURE 51]

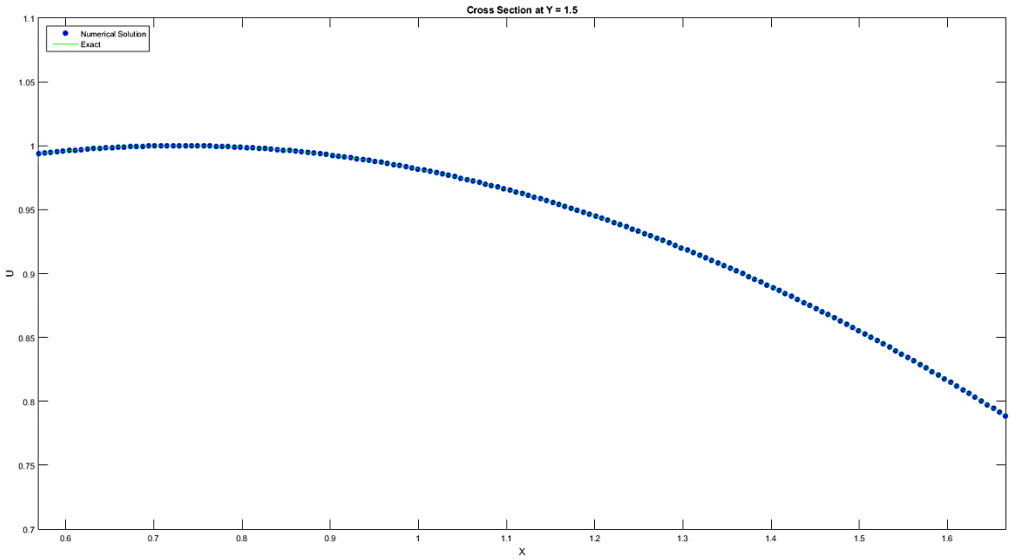


[FIGURE 52]

Contour lines (Figure 51) and a plot of the left-to-right diagonal (Figure 52) with $N = 80$ and $M = 80$.

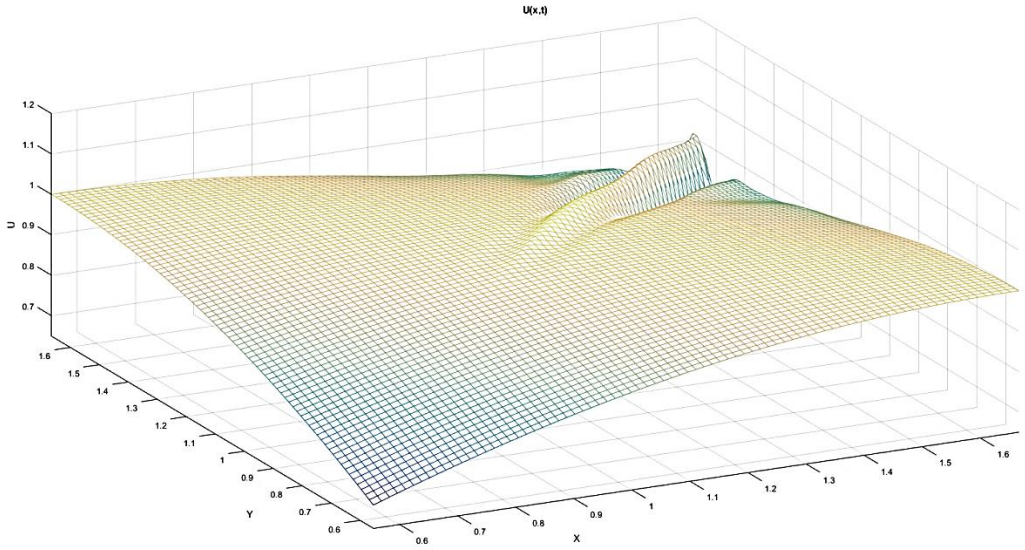


[FIGURE 53]

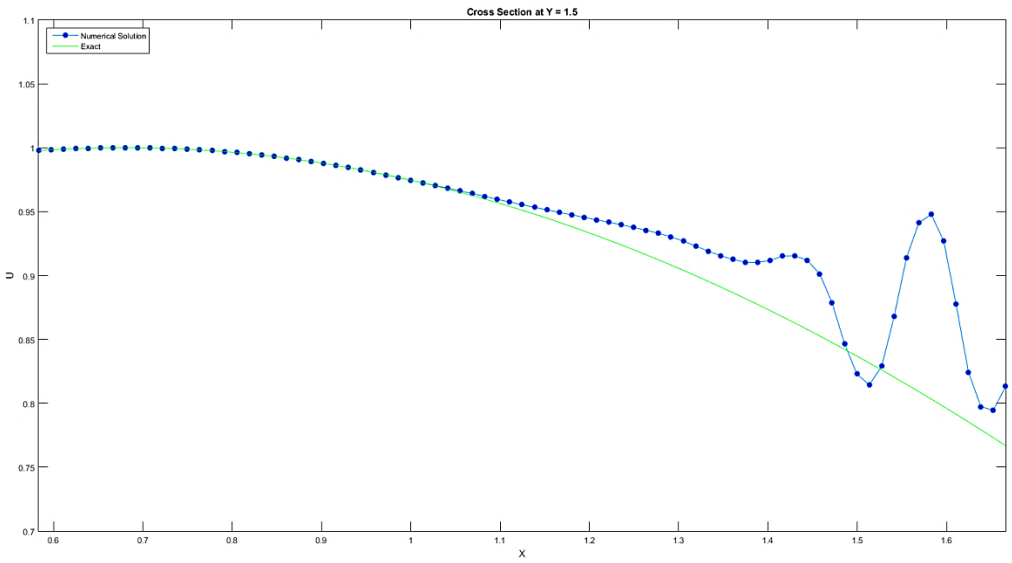


[FIGURE 54]

Various cross-sections at $Y = 0.5, 1.5$ (Figure 53 and 54) with $N = 80$ and $M = 80$.



[FIGURE 55]



[FIGURE 56]

The beginning of divergence due to a CFL condition of 1.1 (Figure 55) at $t = 0.54$, along with a cross-section at $Y = 1.5$ (Figure 56) with $N = 80$ and $M = 80$.

.

Error and Iteration Count Comparison

N	L_∞ error	Order	L_1 error	Order	Time-Marching	Fast Sweeping	CFL $= 1$
40	2.7941e-05	-	9.6513e-06	-	229	211	85
80	1.7496e-06	4.00	7.8847e-07	3.61	429	395	150
160	1.0575e-07	4.05	5.5526e-08	3.83	834	743	275
320	6.2764e-09	4.07	3.0990e-09	4.16	1363	1247	515

[TABLE 12]

For all discretizations, a comparison is made of the error and order for the fast-sweeping and time-marching algorithms, including iteration count. The extrapolation factor is set to 1.

Extrapolation Factor Comparison on Iteration Count

ω	$N = 40$	$N = 80$	$N = 160$
0.9	238	446	839
1.0	211	395	743
1.25	162	301	571
1.5	130	236	447
2	85	150	274
2.1	78	135	243
2.25	-	-	-
Optimal Iteration Reduction %	63.03%	65.82%	67.29%

[TABLE 13]

For all discretizations, various extrapolation factors are used within the fast-sweeping Gauss-Seidel scheme in order to reduce iteration count. The percentage reduction is a comparison of the $\omega = 1$ iteration count and the minimum able to be achieved.

5. CONCLUSION

COMPARISON OF ITERATION OPTIMIZATION

Example 1:

	Time Marching Runge-Kutta WENO				Centered Difference			
CFL	$N = 80$	$N = 160$	$N = 320$	$N = 640$	$N = 80$	$N = 160$	$N = 320$	$N = 640$
0.01	2.1423e-04	1.0932e-04	5.7976e-05	2.9373e-05	0.0404	0.0209	0.0106	0.0054
0.1	0.0022	0.0011	5.8444e-04	2.9416e-04	0.0505	0.0261	0.0133	0.0067
0.5	0.0112	0.0058	0.0029	0.0015	0.0947	0.0490	0.0249	0.0125
1	0.0222	0.0116	0.0059	0.0152	0.1473	0.0768	0.0392	0.0277
2	0.0461	0.0228	0.0165	-	0.2506	0.1306	0.0685	-

[TABLE 14]

Example 2:

N	Time-Marching	Fast Sweeping	Optimization
80	2098	241	88.51%
160	3918	753	80.78%
320	7580	1444	80.95%
640	14446	2954	79.55%

[TABLE 15]

Example 3:

N	Time-Marching	Fast Sweeping	Optimization
80	482	154	70.10%
160	941	256	74.68%
320	1839	510	74.70%
640	3592	1010	75.54%

[TABLE 16]

Example 4:

N	Time-Marching	Fast Sweeping	Optimization
20	152	82	46.05%
40	242	114	52.89%
80	418	179	57.18%
160	767	318	58.54%

[TABLE 17]

Example 5:

N	Time Marching	Fast Sweeping	Optimization
40	975	293	69.95%
80	1828	513	71.94%

[TABLE 18]

Example 6:

N	Time Marching	Fast Sweeping	Optimization
40	229	78	65.94%
80	429	135	68.53%
160	834	243	70.86%

[TABLE 19]

CONCLUDING REMARKS

Across all examples it is clear that the fast sweeping Gauss-Seidel total variation diminishing scheme offers significantly reduced iteration counts to achieve convergence by comparison to its time marching counterpart. Not only does the fast sweeping methodology converge at roughly twice the speed as the time marching approach for Examples 2 and 3, but after adjustments are made to the extrapolation factor, all examples show on average a 50% reduction in iteration count, and for some examples the

reduction is above 80%. In Example 5, convergence was possible at larger discretizations only when the fast sweeping scheme was utilized, showcasing that not only is this scheme more expedient in nature than the time-marching approach, but that it is also more reliable in terms of ensuring overall convergence to the steady state solution. For example 6, where there is no shock or discontinuity, the unmodified fast sweeping method is roughly equivalent in iteration count to the time marching approach, however significant improvements are still possible when the extrapolation factor is increased, according to (Table 11).

On the shock post-processing section of this thesis, an interesting extension might be to consider assigning two separate discretizations to the entire domain, where the area of discontinuity is given a finer mesh than the surrounding smoother regions. Each discretization is uniform [10], and such a scheme could potentially achieve greater accuracy even while user coarser mesh dimensions. Another angle might be that, instead of a post-processing procedure, the smoothness indicator function is continually updating the possible location of discontinuity, and this region is dynamically assigned a more refined mesh, while the remaining regions are given a coarser mesh.

Now, all that has been considered in this thesis has merely been the iteration count for each algorithm to obtain convergence, without concern for the actual running time of the algorithms involved. This is because these algorithms (see Appendix) are notoriously unfriendly for Matlab, and lengthy run times are an unavoidable reality for the 2-D examples with fine mesh discretizations, ranging from several minutes to nearly an entire day. A future extension of these results and the Matlab code would be to enhance the run times of these algorithms, and to therefore make these algorithms desirable both in terms of iteration count and run time.

6. REFERENCES

[1] Bressan, Alberto. “Lecture Notes on Hyperbolic Conservation Laws.” *Department of Mathematics, Penn State University*, 2009

[2] Giordano, Nicholas and Nakanishi, Hisao. “Computational Physics.” Upper Saddle River: Pearson Prentice Hall, 2nd edition, 2006. Print

[3] Strang, Gilbert. “Computational Science and Engineering.” Wellesly: Cambridge, 2007. Print

[4] Burden, Richard and Faires, Douglas and Burden, Annette. “Numerical Analysis.” Boston: Cengage Learning, 10th edition, 2016. Print

[5] Chen, Shanqin. “Fixed-Point Fast Sweeping WENO Methods for Steady State Solutions of Scalar Hyperbolic Conservation Laws.” International Journal of Numerical Analysis and Modeling, Vol. 1, No. 1, 2013, pages 1-15

[6] Shu, Chi-Wang. “Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws.” ICASE Report No. 97-65, 1997, pages 1-78

[7] Shu, Chi-Wang and Jiang, Guang-Shan. “Efficient Implementation of Weighted ENO Schemes.” Journal of Computational Physics, Vol. 126, No. 0130, 1996, pages 202-228

[8] Harten, Ami and Engquist, Bjorn and Osher, Stanley and Chakravarthy, Sukumar. “Uniformly High Order Accurate Essentially Non-oscillatory Schemes, III.” Journal of Computational Physics, Vol. 131, No. CP965632, 1997, pages 3-47

[9] Wu, Liang and Zhang, Yong-Tao and Zhang, Shuhai and Shu, Chi-Wang. “High Order Fixed-Point Sweeping WENO Methods for Steady State of Hyperbolic Conservation Laws and Its Convergence Study.” Communications in Computation Physics, Vol. 20, No. 4, 2016, pages 835-869

[10] Shu, Chi-Wang. “Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws.” In A. Quarteroni, editor, Advanced Numerical Approximation of Nonlinear Hyperbolic Equations, pages 325-432. Lecture Notes in Mathematics, v1697, Springer, 1998.

[11] Levy, Doron and Puppo, Gabriella and Russo, Giovanni. “Central WENO Schemes for Hyperbolic Systems of Conservation Laws.” Mathematical Modelling and Numerical Analysis, Vol. 33 No. 3, 1999, pages 547-571

[12] Zhang, Yong-Tao and Zhao, Hong-Kai and Qian, Jianliang. “High Order Fast Sweeping Methods for Static Hamilton-Jacobi Equations.” Journal of Scientific Computing, 2005

[13] Shu, Chi-Wang and Gottlieb, Sigal. “Total Variation Diminishing Runge-Kutta Schemes.” Mathematics of Computation, Vol. 67, No. 221, 1998, pages 73-85

[14] Chen, Shanqin and Zhang, Yong-Tao and Zhao, Hong-Kai. “Fixed-Point Iterative Sweeping Methods for Static Hamilton-Jacobi Equations.” Methods and Applications of Analysis, Vol. 13, No. 3, 2006, pages 299-320

[15] Wenjie, Wang and Jianhu, Feng and Wei, Xu. “The Numerical Solution of TVD Runge-Kutta and WENO Scheme to the FPK Equations to Nonlinear System of One-Dimension.” Applied and Computational Mathematics, Vol. 5, No. 3, 2016, pages 160-164

[16] Chen, Weitao, and Chou, Ching-Shang, and Kao, Chiu-Yen. “Lax-Friedrich Fast Sweeping Methods for Steady State Problems for Hyperbolic Conservation Laws.” Journal of Computational Physics, Vol. 234, 2013, pages 452-471

[17] Sebastian, Kurt and Shu, Chi-Wang. “Multi Domain WENO Finite Difference Methods with Interpolation at Sub-Domain Interfaces.” Division of Applied Mathematics, Brown University

[18] Rathan, Samala and Raju, G Naga. “An Improved Non-Linear Weight for Seventh-Order WENO Schemes.” Department of Mathematics, Visvesvaraya National Institute of Technology, 2016

[19] Appadu, A.R. and Peer, A. A. I. “Optimized Weighted Essentially Non-Oscillatory Third-Order Schemes for Hyperbolic Conservation Laws.” Journal of Applied Mathematics, Article ID 428681, 2013

[20] Zill, Dennis and Cullen, Michael. “Differential Equations with Boundary-Value Problems.” Belmont: Cengage Learning, 7th edition, 2009. Print

7. APPENDICES

WENO RUNGE-KUTTA ALGORITHM IN PSEUDO-CODE

Given:

Equation

$$U_t + \left(\frac{1}{2}U^2\right)_x = \phi$$

Initial/Boundary conditions

$$U^0 = U(x,0) \qquad U(a,t) = \alpha \qquad U(b,t) = \beta$$

Discretization

$$x \in \{x_1 = a, x_2, x_3, \dots, x_i, \dots, x_{N-2}, x_{N-1}, x_N = b\}$$

While: $L_1\text{-}norm \geq \delta \qquad \delta = 10^{-11}$

Calculate $U^{(1)}$

Determine $\hat{f}^n_{i+\frac{1}{2}}$ and $\hat{f}^n_{i-\frac{1}{2}}$

Determine $L(U_i^n) = -\frac{1}{dx}\left(\hat{f}^n_{i+\frac{1}{2}} - \hat{f}^n_{i-\frac{1}{2}}\right) + \phi_i$

Determine $U_i^{(1)} = U_i^n + dt \cdot L(U_i^n)$

Calculate $U^{(2)}$

Determine $\hat{f}^{(1)}_{i+\frac{1}{2}}$ and $\hat{f}^{(1)}_{i-\frac{1}{2}}$

Determine $L\left(U_i^{(1)}\right) = -\frac{1}{dx}\left(\hat{f}^{(1)}_{i+\frac{1}{2}} - \hat{f}^{(1)}_{i-\frac{1}{2}}\right) + \phi_i$

Determine $U_i^{(2)} = \frac{3}{4}U_i^n + \frac{1}{4}U_i^{(1)} + \frac{1}{4}dt \cdot L(U_i^{(1)})$

Calculate U^{n+1}

Determine $\hat{f}^{(2)}_{i+\frac{1}{2}}$ and $\hat{f}^{(2)}_{i-\frac{1}{2}}$

Determine $L\left(U_i^{(2)}\right) = -\frac{1}{dx}\left(\hat{f}^{(2)}_{i+\frac{1}{2}} - \hat{f}^{(2)}_{i-\frac{1}{2}}\right) + \phi_i$

Determine $U_i^{n+1} = \frac{1}{3}U_i^n + \frac{2}{3}U_i^{(2)} + \frac{2}{3}dt \cdot L(U_i^{(2)})$

Calculate L_1-norm

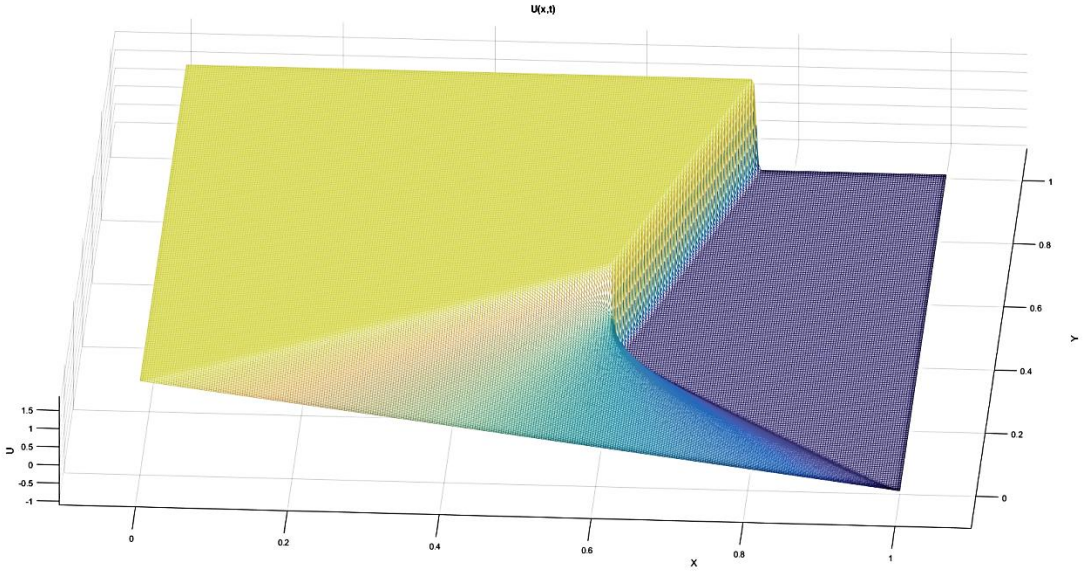
$$L_1 = \frac{1}{N}\sum_{i=1}^{N}\|U^n - U^{n+1}\|$$

Update values

$$U^n = U^{n+1}$$

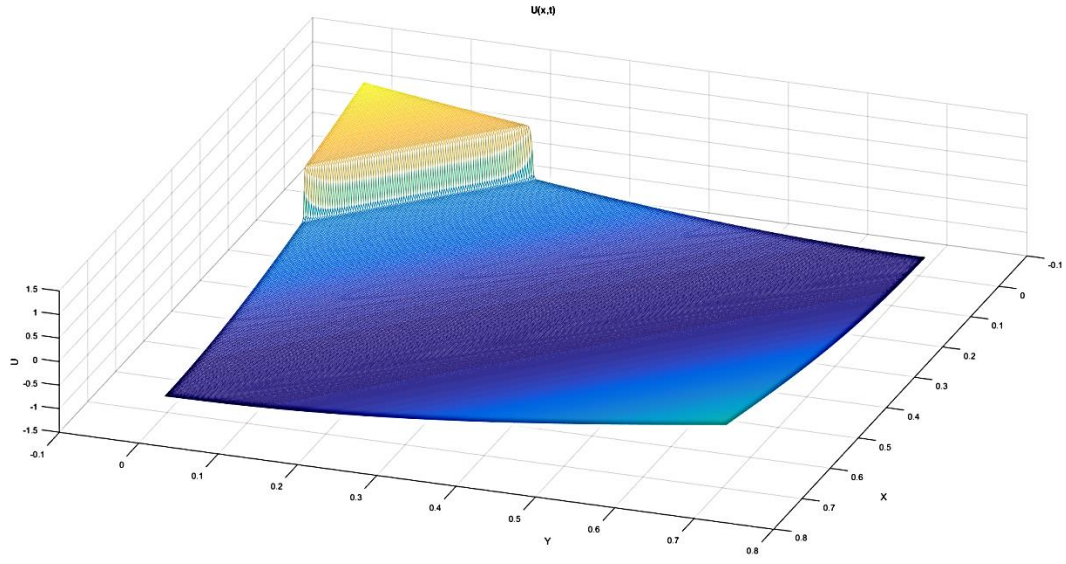
End

ADDITIONAL FIGURES



[FIGURE 57]

Example 4 at steady state on a 320-by-320 grid.



[FIGURE 58]

Example 5 at steady state on a 320-by-320 grid. This graph required over 22 hours of runtime to produce, so I felt obliged to include it in the Appendix section.

MATLAB IMPLEMENTATION OF TIME MARCHING EXAMPLE 3

```
    %Initialize domain
a = 0;
b = pi;
    %Set discretization
N = 80;
DX = (b-a)/N;
X = (a:DX:b);
    %Assign ghost points
X = horzcat(a-DX,X,X(end)+DX);
    %Constants
sigma = 10^-11;
epsilon = 10^-6;
    %Initial conditions
U = 0.5*sin(X);
    %Set boundary conditions for ghost points
U(1) = sin(X(1));
U(end) = -sin(X(end));
    %Initialize U1, U2, and Un+1 arrays
U_1 = [U(1),U(2),zeros(1,numel(X)-4),U(end-1),U(end)];
U_2 = [U(1),U(2),zeros(1,numel(X)-4),U(end-1),U(end)];
U_new = [U(1),U(2),zeros(1,numel(X)-4),U(end-1),U(end)];
    %Initialize the exact solution
exact = zeros(1,numel(X));
    %Assign the exact solution
for k = 1:numel(X)
    if X(k) < (2*pi)/3
        exact(k) = sin(X(k));
    else
        exact(k) = -1*sin(X(k));
    end
end
    %Initialize constants for the while loop
L1norm = inf;
iterations = 0;
    %Iterate until successive iterations fall below
    %the threshold established by sigma
while L1norm > sigma
        %Lax-Friedrich flux and time discretization
    alphaFlux = max(abs(U));
    DT = (1/alphaFlux)*0.5*DX;
        %Iterate through all points, not including ghost points
        %in order to calculate U1
    for i = 3:numel(X)-2
            %Calculate positive flux terms
        fpm2 = 0.5*(0.5*U(i-2)^2 + alphaFlux*U(i-2));
        fpm1 = 0.5*(0.5*U(i-1)^2 + alphaFlux*U(i-1));
        fp = 0.5*(0.5*U(i)^2 + alphaFlux*U(i));
        fpp1 = 0.5*(0.5*U(i+1)^2 + alphaFlux*U(i+1));
        fpp2 = 0.5*(0.5*U(i+2)^2 + alphaFlux*U(i+2));
```

```
            %Calculate negative flux terms
        fmm2 = 0.5*(0.5*U(i-2)^2 - alphaFlux*U(i-2));
        fmm1 = 0.5*(0.5*U(i-1)^2 - alphaFlux*U(i-1));
        fm = 0.5*(0.5*U(i)^2 - alphaFlux*U(i));
        fmp1 = 0.5*(0.5*U(i+1)^2 - alphaFlux*U(i+1));
        fmp2 = 0.5*(0.5*U(i+2)^2 - alphaFlux*U(i+2));
            %Calculate positive flux terms at i+1/2 position
        b0 = (fpp1 - fp)^2;
        b1 = (fp - fpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpp = w0*(0.5*fp + 0.5*fpp1) + w1*(-0.5*fpm1 + (3/2)*fp);
            %Calculate positive flux terms at i-1/2 position
        b0 = (fp - fpm1)^2;
        b1 = (fpm1 - fpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpm = w0*(0.5*fpm1 + 0.5*fp) + w1*(-0.5*fpm2 + (3/2)*fpm1);
            %Calculate negative flux terms at i+1/2 position
        b0 = (fmp2 - fmp1)^2;
        b1 = (fmp1 - fm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmp = w0*((3/2)*fmp1 - 0.5*fmp2) + w1*(0.5*fm + 0.5*fmp1);
            %Calculate negative flux terms at i-1/2 position
        b0 = (fmp1 - fm)^2;
        b1 = (fm - fmm1)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmm = w0*((3/2)*fm - 0.5*fmp1) + w1*(0.5*fmm1 + 0.5*fm);
            %Combine terms
        fhp = fpp + fmp;
        fhm = fpm + fmm;
            %Calculate L(U)
        LUn_t = (-1/DX)*(fhp - fhm) + sin(X(i))*cos(X(i));
            %Calculate U1
        U_1(i) = U(i) + DT*LUn_t;
    end
        %Iterate through all points, not including ghost points
        %in order to calculate U2
        for i = 3:numel(X)-2
            %Calculate positive flux terms
        fpm2 = 0.5*(0.5*U_1(i-2)^2 + alphaFlux*U_1(i-2));
        fpm1 = 0.5*(0.5*U_1(i-1)^2 + alphaFlux*U_1(i-1));
        fp = 0.5*(0.5*U_1(i)^2 + alphaFlux*U_1(i));
```

```
        fpp1 = 0.5*(0.5*U_1(i+1)^2 + alphaFlux*U_1(i+1));
        fpp2 = 0.5*(0.5*U_1(i+2)^2 + alphaFlux*U_1(i+2));
            %Calculate negative flux terms
        fmm2 = 0.5*(0.5*U_1(i-2)^2 - alphaFlux*U_1(i-2));
        fmm1 = 0.5*(0.5*U_1(i-1)^2 - alphaFlux*U_1(i-1));
        fm = 0.5*(0.5*U_1(i)^2 - alphaFlux*U_1(i));
        fmp1 = 0.5*(0.5*U_1(i+1)^2 - alphaFlux*U_1(i+1));
        fmp2 = 0.5*(0.5*U_1(i+2)^2 - alphaFlux*U_1(i+2));
            %Calculate positive flux terms at i+1/2 position
        b0 = (fpp1 - fp)^2;
        b1 = (fp - fpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpp = w0*(0.5*fp + 0.5*fpp1) + w1*(-0.5*fpm1 + (3/2)*fp);
            %Calculate positive flux terms at i-1/2 position
        b0 = (fp - fpm1)^2;
        b1 = (fpm1 - fpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpm = w0*(0.5*fpm1 + 0.5*fp) + w1*(-0.5*fpm2 + (3/2)*fpm1);
            %Calculate negative flux terms at i+1/2 position
        b0 = (fmp2 - fmp1)^2;
        b1 = (fmp1 - fm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmp = w0*((3/2)*fmp1 - 0.5*fmp2) + w1*(0.5*fm + 0.5*fmp1);
            %Calculate negative flux terms at i-1/2 position
        b0 = (fmp1 - fm)^2;
        b1 = (fm - fmm1)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmm = w0*((3/2)*fm - 0.5*fmp1) + w1*(0.5*fmm1 + 0.5*fm);
            %Combine terms
        fhp = fpp + fmp;
        fhm = fpm + fmm;
            %Calculate L(U1)
        LU1_tDt = (-1/DX)*(fhp - fhm) + sin(X(i))*cos(X(i));
            %Calculate U2
        U_2(i) = (3/4)*U(i) + (1/4)*U_1(i) + (1/4)*(DT)*LU1_tDt;
    end
    %Iterate through all points, not including ghost points
    %in order to calculate Un+1
    for i = 3:numel(X)-2
        %Calculate positive flux terms
        fpm2 = 0.5*(0.5*U_2(i-2)^2 + alphaFlux*U_2(i-2));
```

```
        fpm1 = 0.5*(0.5*U_2(i-1)^2 + alphaFlux*U_2(i-1));
        fp = 0.5*(0.5*U_2(i)^2 + alphaFlux*U_2(i));
        fpp1 = 0.5*(0.5*U_2(i+1)^2 + alphaFlux*U_2(i+1));
        fpp2 = 0.5*(0.5*U_2(i+2)^2 + alphaFlux*U_2(i+2));
            %Calculate negative flux terms
        fmm2 = 0.5*(0.5*U_2(i-2)^2 - alphaFlux*U_2(i-2));
        fmm1 = 0.5*(0.5*U_2(i-1)^2 - alphaFlux*U_2(i-1));
        fm = 0.5*(0.5*U_2(i)^2 - alphaFlux*U_2(i));
        fmp1 = 0.5*(0.5*U_2(i+1)^2 - alphaFlux*U_2(i+1));
        fmp2 = 0.5*(0.5*U_2(i+2)^2 - alphaFlux*U_2(i+2));
            %Calculate positive flux terms at i+1/2 position
        b0 = (fpp1 - fp)^2;
        b1 = (fp - fpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpp = w0*(0.5*fp + 0.5*fpp1) + w1*(-0.5*fpm1 + (3/2)*fp);
            %Calculate positive flux terms at i-1/2 position
        b0 = (fp - fpm1)^2;
        b1 = (fpm1 - fpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpm = w0*(0.5*fpm1 + 0.5*fp) + w1*(-0.5*fpm2 + (3/2)*fpm1);
            %Calculate negative flux terms at i+1/2 position
        b0 = (fmp2 - fmp1)^2;
        b1 = (fmp1 - fm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmp = w0*((3/2)*fmp1 - 0.5*fmp2) + w1*(0.5*fm + 0.5*fmp1);
            %Calculate negative flux terms at i-1/2 position
        b0 = (fmp1 - fm)^2;
        b1 = (fm - fmm1)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmm = w0*((3/2)*fm - 0.5*fmp1) + w1*(0.5*fmm1 + 0.5*fm);
            %Combine terms
        fhp = fpp + fmp;
        fhm = fpm + fmm;
            %Calculate L(U2)
        LU2_thDt = (-1/DX)*(fhp - fhm) + sin(X(i))*cos(X(i));
            %Calculate Un+1
        U_new(i) = (1/3)*U(i) + (2/3)*U_2(i) + (2/3)*(DT)*LU2_thDt;
       end
        %Calculate L1 norm of successive iterations
    L1norm = max(abs(U_new - U));
        %Update iteration count
```

```
    iterations = iterations + 1;
        %Update U values
    U = U_new;
end
    %Output iterations
iterations
    %Initialize exact and U variants without shock values
exact_m = exact;
U_m = U;
    %Shock location
c = 2*pi/3;
for j = 1:numel(X)
    if (X(j) > c - 0.1) && (X(j) < c + 0.1)
            %If the value of X is within 0.1 of the shock
            %then zero out the values of exact and U variants
        exact_m(j) = 0;
        U_m(j) = 0;
    end
end
    %Calculate L1 and Linfinity errors
L_inf = norm(U_m - exact_m,inf)
L_1 = (1/numel(U_m))*norm(U_m - exact_m,1)
```

Published with MATLAB® R2015a

MATLAB IMPLEMENTATION OF FAST SWEEPING EXAMPLE 6

```
    %Initialize domain
a = pi/(4*sqrt(2));
b = 3*pi/(4*sqrt(2));
    %Set discretization
N = 40;
DX = (b-a)/N;
DY = DX;
X = (a:DX:b);
    %Assign ghost points
X = horzcat(a-2*DX,a-DX,X,X(end)+DX,X(end)+2*DX);
    %Same assignment for Y
Y = X;
    %Constants
sigma = 10^-11;
epsilon = 10^-6;
gamma = 0.5;
omega = 1;
    %Initialize solution and exact arrays
U = zeros(numel(X),numel(Y));
exact = zeros(numel(X),numel(Y));
for i = 1:numel(Y)
    for j = 1:numel(X)
            %Initial conditions
        U(j,i) = 1.5*sin((X(i) + Y(j))/(sqrt(2)));
            %Initialize the exact solution
        exact(j,i) = sin((X(i) + Y(j))/(sqrt(2)));
    end
end
    %Create arrays for the index values of X and Y
listx = 1:1:numel(X);
listy = 1:1:numel(Y);
    %Set boundary conditions for ghost points
U(1,:) = exact(1,:);
U(2,:) = exact(2,:);
U(end,:) = exact(end,:);
U(end-1,:) = exact(end-1,:);
U(:,1) = exact(:,1);
U(:,2) = exact(:,2);
U(:,end) = exact(:,end);
U(:,end-1) = exact(:,end-1);
    %Initialize U1, U2, and Un+1 arrays
U_1 = U;
U_2 = U;
U_new = U;
    %Initialize constants for the while loop
L1norm = inf;
iterations = 0;
    %Initialize the variable that informs the sweep direction
sweep_direction = 0;
```

```
    %The direction variable will rotate among the values
    %1,2,3, and 0 which specifies the sweep direction
direction = mod(sweep_direction,4);
    %Iterate until successive iterations fall below
    %the threshold established by sigma
while L1norm > sigma && iterations < 2000
        %Reset the list of index values
    listx = 1:1:numel(X);
    listy = 1:1:numel(Y);
        %Lax-Friedrich flux and time discretization
    term = max(max(U));
    alphaFlux_X = max(max(abs( (1/sqrt(2))*term^2)));
    alphaFlux_Y = max(max(abs( (1/sqrt(2))*term^2)));
    DT = gamma*DX*(1/alphaFlux_X);
        %Update sweep direction
    sweep_direction = sweep_direction + 1;
        %Apply modulus to this value
    direction = mod(sweep_direction,4);
        %Determine the indexing according to the direction variable
    if direction == 1
        listx = listx;
        listy = listy;
    elseif direction == 2
        listx = flip(listx);
        listy = listy;
    elseif direction == 3
        listx = flip(listx);
        listy = flip(listy);
    elseif direction == 0
        listx = listx;
        listy = flip(listy);
    end
        %Reset the number of established points this iteration
    x_points = 0;
    y_points = 0;
        %Iterate through all points, not including ghost points
        %in order to calculate U1
    for i = listx(3:end-2);
        for j = listy(3:end-2);
                %Y direction
                %Determine stencil parameters
            if y_points == 0
                    %When there are no available new values
                A = U(j-2,i);
                B = U(j-1,i);
                C = U(j,i);
                D = U(j+1,i);
                E = U(j+2,i);
            elseif y_points == 1 && (direction == 1 || direction == 2)
                    %When there is 1 available new value and
                    %the sweep direction is 1:M
                A = U(j-2,i);
                B = U_1(j-1,i);
```

```
            C = U(j,i);
            D = U(j+1,i);
            E = U(j+2,i);
        elseif y_points >= 2 && (direction == 1 || direction == 2)
                %When there is 2 or more available new values and
                %the sweep direction is 1:M
            A = U_1(j-2,i);
            B = U_1(j-1,i);
            C = U(j,i);
            D = U(j+1,i);
            E = U(j+2,i);
        elseif y_points == 1 && (direction == 3 || direction == 0)
                %When there is 1 available new value and
                %the sweep direction is M:1
            A = U(j-2,i);
            B = U(j-1,i);
            C = U(j,i);
            D = U_1(j+1,i);
            E = U(j+2,i);
        elseif y_points >= 2 && (direction == 3 || direction == 0)
                %When there is 2 or more available new values and
                %the sweep direction is M:1
            A = U(j-2,i);
            B = U(j-1,i);
            C = U(j,i);
            D = U_1(j+1,i);
            E = U_1(j+2,i);
        end
            %Calculate positive flux terms
        gpm2 = 0.5*(0.5*(1/sqrt(2))*A^2 + alphaFlux_Y*A);
        gpm1 = 0.5*(0.5*(1/sqrt(2))*B^2 + alphaFlux_Y*B);
        gp = 0.5*(0.5*(1/sqrt(2))*C^2 + alphaFlux_Y*C);
        gpp1 = 0.5*(0.5*(1/sqrt(2))*D^2 + alphaFlux_Y*D);
        gpp2 = 0.5*(0.5*(1/sqrt(2))*E^2 + alphaFlux_Y*E);
            %Calculate negative flux terms
        gmm2 = 0.5*(0.5*(1/sqrt(2))*A^2 - alphaFlux_Y*A);
        gmm1 = 0.5*(0.5*(1/sqrt(2))*B^2 - alphaFlux_Y*B);
        gm = 0.5*(0.5*(1/sqrt(2))*C^2 - alphaFlux_Y*C);
        gmp1 = 0.5*(0.5*(1/sqrt(2))*D^2 - alphaFlux_Y*D);
        gmp2 = 0.5*(0.5*(1/sqrt(2))*E^2 - alphaFlux_Y*E);
            %Calculate positive flux terms at i+1/2 position
        b0 = (gpp1 - gp)^2;
        b1 = (gp - gpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        gpp = w0*(0.5*gp + 0.5*gpp1) + w1*(-0.5*gpm1 + (3/2)*gp);
            %Calculate positive flux terms at i-1/2 position
        b0 = (gp - gpm1)^2;
        b1 = (gpm1 - gpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
```

```
w0 = a0/(a0 + a1);
w1 = a1/(a0 + a1);
gpm = w0*(0.5*gpm1 + 0.5*gp) + w1*(-0.5*gpm2 + (3/2)*gpm1);
    %Calculate negative flux terms at i+1/2 position
b0 = (gmp2 - gmp1)^2;
b1 = (gmp1 - gm)^2;
a0 = (1/3)/(epsilon + b0)^2;
a1 = (2/3)/(epsilon + b1)^2;
w0 = a0/(a0 + a1);
w1 = a1/(a0 + a1);
gmp = w0*((3/2)*gmp1 - 0.5*gmp2) + w1*(0.5*gm + 0.5*gmp1);
    %Calculate negative flux terms at i-1/2 position
b0 = (gmp1 - gm)^2;
b1 = (gm - gmm1)^2;
a0 = (1/3)/(epsilon + b0)^2;
a1 = (2/3)/(epsilon + b1)^2;
w0 = a0/(a0 + a1);
w1 = a1/(a0 + a1);
gmm = w0*((3/2)*gm - 0.5*gmp1) + w1*(0.5*gmm1 + 0.5*gm);
    %Combine terms
ghp = gpp + gmp;
ghm = gpm + gmm;

    %X direction
    %Determine stencil parameters
if x_points == 0
        %When there are no available new values
    A = U(j,i-2);
    B = U(j,i-1);
    C = U(j,i);
    D = U(j,i+1);
    E = U(j,i+2);
elseif x_points == 1 && (direction == 1 || direction == 0)
        %When there is 1 available new value and
        %the sweep direction is 1:N
    A = U(j,i-2);
    B = U_1(j,i-1);
    C = U(j,i);
    D = U(j,i+1);
    E = U(j,i+2);
elseif x_points >= 2 && (direction == 1 || direction == 0)
        %When there is 2 or more available new values and
        %the sweep direction is 1:N
    A = U_1(j,i-2);
    B = U_1(j,i-1);
    C = U(j,i);
    D = U(j,i+1);
    E = U(j,i+2);
elseif x_points == 1 && (direction == 2 || direction == 3)
        %When there is 1 available new value and
        %the sweep direction is N:1
    A = U(j,i-2);
    B = U(j,i-1);
```

```
            C = U(j,i);
            D = U_1(j,i+1);
            E = U(j,i+2);
        elseif x_points >= 2 && (direction == 2 || direction == 3)
                %When there is 2 or more available new values and
                %the sweep direction is N:1
            A = U(j,i-2);
            B = U(j,i-1);
            C = U(j,i);
            D = U_1(j,i+1);
            E = U_1(j,i+2);
        end
            %Calculate positive flux terms
        fpm2 = 0.5*(0.5*(1/sqrt(2))*A^2 + alphaFlux_X*A);
        fpm1 = 0.5*(0.5*(1/sqrt(2))*B^2 + alphaFlux_X*B);
        fp = 0.5*(0.5*(1/sqrt(2))*C^2 + alphaFlux_X*C);
        fpp1 = 0.5*(0.5*(1/sqrt(2))*D^2 + alphaFlux_X*D);
        fpp2 = 0.5*(0.5*(1/sqrt(2))*E^2 + alphaFlux_X*E);
            %Calculate negative flux terms
        fmm2 = 0.5*(0.5*(1/sqrt(2))*A^2 - alphaFlux_X*A);
        fmm1 = 0.5*(0.5*(1/sqrt(2))*B^2 - alphaFlux_X*B);
        fm = 0.5*(0.5*(1/sqrt(2))*C^2 - alphaFlux_X*C);
        fmp1 = 0.5*(0.5*(1/sqrt(2))*D^2 - alphaFlux_X*D);
        fmp2 = 0.5*(0.5*(1/sqrt(2))*E^2 - alphaFlux_X*E);
            %Calculate positive flux terms at i+1/2 position
        b0 = (fpp1 - fp)^2;
        b1 = (fp - fpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpp = w0*(0.5*fp + 0.5*fpp1) + w1*(-0.5*fpm1 + (3/2)*fp);
            %Calculate positive flux terms at i-1/2 position
        b0 = (fp - fpm1)^2;
        b1 = (fpm1 - fpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpm = w0*(0.5*fpm1 + 0.5*fp) + w1*(-0.5*fpm2 + (3/2)*fpm1);
            %Calculate negative flux terms at i+1/2 position
        b0 = (fmp2 - fmp1)^2;
        b1 = (fmp1 - fm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmp = w0*((3/2)*fmp1 - 0.5*fmp2) + w1*(0.5*fm + 0.5*fmp1);
            %Calculate negative flux terms at i-1/2 position
        b0 = (fmp1 - fm)^2;
        b1 = (fm - fmm1)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
```

```
            w0 = a0/(a0 + a1);
            w1 = a1/(a0 + a1);
            fmm = w0*((3/2)*fm - 0.5*fmp1) + w1*(0.5*fmm1 + 0.5*fm);
                %Combine terms
            fhp = fpp + fmp;
            fhm = fpm + fmm;
            term = X(i) + Y(j);
            term = pi*term;
            term = term/sqrt(2);
                %Calculate L
            L = -((1/DX)*(fhp - fhm)) - ((1/DY)*(ghp - ghm)) + ...
                (sin((X(i) + Y(j))/(sqrt(2))))*(cos((X(i) + Y(j))/(sqrt(2))));
                %Calculate U1
            U_1(j,i) = U(j,i) + gamma/((alphaFlux_X/DX) + (alphaFlux_X/DY))*L;
                %Calculate U1 with extrapolation factor
            U_1(j,i)= omega*U_1(j,i) + (1-omega)*U(j,i);
                %Increase the number of new Y values
            y_points = y_points + 1;
        end
            %Increase the number of new X values
        x_points = x_points + 1;
            %Reset the number of Y values
        y_points = 0;
    end
        %Reset both X and Y values
    x_points = 0;
    y_points = 0;
        %Iterate through all points, not including ghost points
        %in order to calculate U2
    for i = listx(3:end-2);
        for j = listy(3:end-2);
                %Y direction
                %Determine stencil parameters
            if y_points == 0
                    %When there are no available new values
                A = U_1(j-2,i);
                B = U_1(j-1,i);
                C = U_1(j,i);
                D = U_1(j+1,i);
                E = U_1(j+2,i);
            elseif y_points == 1 && (direction == 1 || direction == 2)
                    %When there is 1 available new value and
                    %the sweep direction is 1:M
                A = U_1(j-2,i);
                B = U_2(j-1,i);
                C = U_1(j,i);
                D = U_1(j+1,i);
                E = U_1(j+2,i);
            elseif y_points >= 2 && (direction == 1 || direction == 2)
                    %When there is 2 or more available new values and
                    %the sweep direction is 1:M
                A = U_2(j-2,i);
                B = U_2(j-1,i);
```

```
            C = U_1(j,i);
            D = U_1(j+1,i);
            E = U_1(j+2,i);
        elseif y_points == 1 && (direction == 3 || direction == 0)
                %When there is 1 available new value and
                %the sweep direction is M:1
            A = U_1(j-2,i);
            B = U_1(j-1,i);
            C = U_1(j,i);
            D = U_2(j+1,i);
            E = U_1(j+2,i);
        elseif y_points >= 2 && (direction == 3 || direction == 0)
                %When there is 2 or more available new values and
                %the sweep direction is M:1
            A = U_1(j-2,i);
            B = U_1(j-1,i);
            C = U_1(j,i);
            D = U_2(j+1,i);
            E = U_2(j+2,i);
        end
            %Calculate positive flux terms
        gpm2 = 0.5*(0.5*(1/sqrt(2))*A^2 + alphaFlux_X*A);
        gpm1 = 0.5*(0.5*(1/sqrt(2))*B^2 + alphaFlux_X*B);
        gp = 0.5*(0.5*(1/sqrt(2))*C^2 + alphaFlux_X*C);
        gpp1 = 0.5*(0.5*(1/sqrt(2))*D^2 + alphaFlux_X*D);
        gpp2 = 0.5*(0.5*(1/sqrt(2))*E^2 + alphaFlux_X*E);
            %Calculate negative flux terms
        gmm2 = 0.5*(0.5*(1/sqrt(2))*A^2 - alphaFlux_X*A);
        gmm1 = 0.5*(0.5*(1/sqrt(2))*B^2 - alphaFlux_X*B);
        gm = 0.5*(0.5*(1/sqrt(2))*C^2 - alphaFlux_X*C);
        gmp1 = 0.5*(0.5*(1/sqrt(2))*D^2 - alphaFlux_X*D);
        gmp2 = 0.5*(0.5*(1/sqrt(2))*E^2 - alphaFlux_X*E);
            %Calculate positive flux terms at i+1/2 position
        b0 = (gpp1 - gp)^2;
        b1 = (gp - gpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        gpp = w0*(0.5*gp + 0.5*gpp1) + w1*(-0.5*gpm1 + (3/2)*gp);
            %Calculate positive flux terms at i-1/2 position
        b0 = (gp - gpm1)^2;
        b1 = (gpm1 - gpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        gpm = w0*(0.5*gpm1 + 0.5*gp) + w1*(-0.5*gpm2 + (3/2)*gpm1);
            %Calculate negative flux terms at i+1/2 position
        b0 = (gmp2 - gmp1)^2;
        b1 = (gmp1 - gm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
```

```
w0 = a0/(a0 + a1);
w1 = a1/(a0 + a1);
gmp = w0*((3/2)*gmp1 - 0.5*gmp2) + w1*(0.5*gm + 0.5*gmp1);
    %Calculate negative flux terms at i-1/2 position
b0 = (gmp1 - gm)^2;
b1 = (gm - gmm1)^2;
a0 = (1/3)/(epsilon + b0)^2;
a1 = (2/3)/(epsilon + b1)^2;
w0 = a0/(a0 + a1);
w1 = a1/(a0 + a1);
gmm = w0*((3/2)*gm - 0.5*gmp1) + w1*(0.5*gmm1 + 0.5*gm);
    %Combine terms
ghp = gpp + gmp;
ghm = gpm + gmm;

%X direction
%Determine stencil parameters
if x_points == 0
        %When there are no available new values
    A = U_1(j,i-2);
    B = U_1(j,i-1);
    C = U_1(j,i);
    D = U_1(j,i+1);
    E = U_1(j,i+2);
elseif x_points == 1 && (direction == 1 || direction == 0)
        %When there is 1 available new value and
        %the sweep direction is 1:N
    A = U_1(j,i-2);
    B = U_2(j,i-1);
    C = U_1(j,i);
    D = U_1(j,i+1);
    E = U_1(j,i+2);
elseif x_points >= 2 && (direction == 1 || direction == 0)
        %When there is 2 or more available new values and
        %the sweep direction is 1:N
    A = U_2(j,i-2);
    B = U_2(j,i-1);
    C = U_1(j,i);
    D = U_1(j,i+1);
    E = U_1(j,i+2);
elseif x_points == 1 && (direction == 2 || direction == 3)
        %When there is 1 available new value and
        %the sweep direction is N:1
    A = U_1(j,i-2);
    B = U_1(j,i-1);
    C = U_1(j,i);
    D = U_2(j,i+1);
    E = U_1(j,i+2);
elseif x_points >= 2 && (direction == 2 || direction == 3)
        %When there is 2 or more available new values and
        %the sweep direction is N:1
    A = U_1(j,i-2);
    B = U_1(j,i-1);
```

```
            C = U_1(j,i);
            D = U_2(j,i+1);
            E = U_2(j,i+2);
        end
            %Calculate positive flux terms
        fpm2 = 0.5*(0.5*(1/sqrt(2))*A^2 + alphaFlux_X*A);
        fpm1 = 0.5*(0.5*(1/sqrt(2))*B^2 + alphaFlux_X*B);
        fp = 0.5*(0.5*(1/sqrt(2))*C^2 + alphaFlux_X*C);
        fpp1 = 0.5*(0.5*(1/sqrt(2))*D^2 + alphaFlux_X*D);
        fpp2 = 0.5*(0.5*(1/sqrt(2))*E^2 + alphaFlux_X*E);
           %Calculate negative flux terms
        fmm2 = 0.5*(0.5*(1/sqrt(2))*A^2 - alphaFlux_X*A);
        fmm1 = 0.5*(0.5*(1/sqrt(2))*B^2 - alphaFlux_X*B);
        fm = 0.5*(0.5*(1/sqrt(2))*C^2 - alphaFlux_X*C);
        fmp1 = 0.5*(0.5*(1/sqrt(2))*D^2 - alphaFlux_X*D);
        fmp2 = 0.5*(0.5*(1/sqrt(2))*E^2 - alphaFlux_X*E);
            %Calculate positive flux terms at i+1/2 position
        b0 = (fpp1 - fp)^2;
        b1 = (fp - fpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpp = w0*(0.5*fp + 0.5*fpp1) + w1*(-0.5*fpm1 + (3/2)*fp);
           %Calculate positive flux terms at i-1/2 position
        b0 = (fp - fpm1)^2;
        b1 = (fpm1 - fpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpm = w0*(0.5*fpm1 + 0.5*fp) + w1*(-0.5*fpm2 + (3/2)*fpm1);
           %Calculate negative flux terms at i+1/2 position
        b0 = (fmp2 - fmp1)^2;
        b1 = (fmp1 - fm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmp = w0*((3/2)*fmp1 - 0.5*fmp2) + w1*(0.5*fm + 0.5*fmp1);
           %Calculate negative flux terms at i-1/2 position
        b0 = (fmp1 - fm)^2;
        b1 = (fm - fmm1)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmm = w0*((3/2)*fm - 0.5*fmp1) + w1*(0.5*fmm1 + 0.5*fm);
            %Combine terms
        fhp = fpp + fmp;
        fhm = fpm + fmm;
        term = X(i) + Y(j);
        term = pi*term;
```

```
            term = term/sqrt(2);
                %Calculate L
            L = -((1/DX)*(fhp - fhm)) - ((1/DY)*(ghp - ghm)) + ...
                (sin((X(i) + Y(j))/(sqrt(2))))*(cos((X(i) + Y(j))/(sqrt(2))));
                %Calculate U2
            U_2(j,i) = U_1(j,i) + (gamma/4)*(1/((alphaFlux_X/DX) + (alphaFlux_X/DY) ))*L;
                %Calculate L with extrapolation factor
            U_2(j,i) = omega*U_2(j,i) + (1-omega)*U_1(j,i);
                %Increase the number of new Y values
            y_points = y_points + 1;
        end
            %Increase the number of new X values
        x_points = x_points + 1;
            %Reset the number of Y values
        y_points = 0;
    end
        %Reset both X and Y values
    x_points = 0;
    y_points = 0;
        %Iterate through all points, not including ghost points
        %in order to calculate Un+1
for i = listx(3:end-2);
       for j = listy(3:end-2);
        %Y direction
        %Determine stencil parameters
        if y_points == 0
                %When there are no available new values
            A = U_2(j-2,i);
            B = U_2(j-1,i);
            C = U_2(j,i);
            D = U_2(j+1,i);
            E = U_2(j+2,i);
        elseif y_points == 1 && (direction == 1 || direction == 2)
                %When there is 1 available new value and
                %the sweep direction is 1:M
            A = U_2(j-2,i);
            B = U_new(j-1,i);
            C = U_2(j,i);
            D = U_2(j+1,i);
            E = U_2(j+2,i);
        elseif y_points >= 2 && (direction == 1 || direction == 2)
                %When there is 2 or more available new values and
                %the sweep direction is 1:M
            A = U_new(j-2,i);
            B = U_new(j-1,i);
            C = U_2(j,i);
            D = U_2(j+1,i);
            E = U_2(j+2,i);
        elseif y_points == 1 && (direction == 3 || direction == 0)
                %When there is 1 available new value and
                %the sweep direction is M:1
            A = U_2(j-2,i);
            B = U_2(j-1,i);
```

```
            C = U_2(j,i);
            D = U_new(j+1,i);
            E = U_2(j+2,i);
        elseif y_points >= 2 && (direction == 3 || direction == 0)
                %When there is 2 or more available new values and
                %the sweep direction is M:1
            A = U_2(j-2,i);
            B = U_2(j-1,i);
            C = U_2(j,i);
            D = U_new(j+1,i);
            E = U_new(j+2,i);
        end
            %Calculate positive flux terms
        gpm2 = 0.5*(0.5*(1/sqrt(2))*A^2 + alphaFlux_X*A);
        gpm1 = 0.5*(0.5*(1/sqrt(2))*B^2 + alphaFlux_X*B);
        gp = 0.5*(0.5*(1/sqrt(2))*C^2 + alphaFlux_X*C);
        gpp1 = 0.5*(0.5*(1/sqrt(2))*D^2 + alphaFlux_X*D);
        gpp2 = 0.5*(0.5*(1/sqrt(2))*E^2 + alphaFlux_X*E);
            %Calculate negative flux terms
        gmm2 = 0.5*(0.5*(1/sqrt(2))*A^2 - alphaFlux_X*A);
        gmm1 = 0.5*(0.5*(1/sqrt(2))*B^2 - alphaFlux_X*B);
        gm = 0.5*(0.5*(1/sqrt(2))*C^2 - alphaFlux_X*C);
        gmp1 = 0.5*(0.5*(1/sqrt(2))*D^2 - alphaFlux_X*D);
        gmp2 = 0.5*(0.5*(1/sqrt(2))*E^2 - alphaFlux_X*E);
               %Calculate positive flux terms at i+1/2 position
        b0 = (gpp1 - gp)^2;
        b1 = (gp - gpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        gpp = w0*(0.5*gp + 0.5*gpp1) + w1*(-0.5*gpm1 + (3/2)*gp);
               %Calculate positive flux terms at i-1/2 position
        b0 = (gp - gpm1)^2;
        b1 = (gpm1 - gpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        gpm = w0*(0.5*gpm1 + 0.5*gp) + w1*(-0.5*gpm2 + (3/2)*gpm1);
               %Calculate negative flux terms at i+1/2 position
        b0 = (gmp2 - gmp1)^2;
        b1 = (gmp1 - gm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        gmp = w0*((3/2)*gmp1 - 0.5*gmp2) + w1*(0.5*gm + 0.5*gmp1);
            %Calculate negative flux terms at i-1/2 position
        b0 = (gmp1 - gm)^2;
        b1 = (gm - gmm1)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
```

```
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        gmm = w0*((3/2)*gm - 0.5*gmp1) + w1*(0.5*gmm1 + 0.5*gm);
               %Combine terms
        ghp = gpp + gmp;
        ghm = gpm + gmm;

        %X direction
        %Determine stencil parameters
        if x_points == 0
                %When there are no available new values
            A = U_2(j,i-2);
            B = U_2(j,i-1);
            C = U_2(j,i);
            D = U_2(j,i+1);
            E = U_2(j,i+2);
        elseif x_points == 1 && (direction == 1 || direction == 0)
                %When there is 1 available new value and
                %the sweep direction is 1:N
            A = U_2(j,i-2);
            B = U_new(j,i-1);
            C = U_2(j,i);
            D = U_2(j,i+1);
            E = U_2(j,i+2);
        elseif x_points >= 2 && (direction == 1 || direction == 0)
                %When there is 2 or more available new values and
                %the sweep direction is 1:N
            A = U_new(j,i-2);
            B = U_new(j,i-1);
            C = U_2(j,i);
            D = U_2(j,i+1);
            E = U_2(j,i+2);
        elseif x_points == 1 && (direction == 2 || direction == 3)
                %When there is 1 available new value and
                %the sweep direction is N:1
            A = U_2(j,i-2);
            B = U_2(j,i-1);
            C = U_2(j,i);
            D = U_new(j,i+1);
            E = U_2(j,i+2);
        elseif x_points >= 2 && (direction == 2 || direction == 3)
                %When there is 2 or more available new values and
                %the sweep direction is N:1
            A = U_2(j,i-2);
            B = U_2(j,i-1);
            C = U_2(j,i);
            D = U_new(j,i+1);
            E = U_new(j,i+2);
        end
               %Calculate positive flux terms
        fpm2 = 0.5*(0.5*(1/sqrt(2))*A^2 + alphaFlux_X*A);
        fpm1 = 0.5*(0.5*(1/sqrt(2))*B^2 + alphaFlux_X*B);
        fp = 0.5*(0.5*(1/sqrt(2))*C^2 + alphaFlux_X*C);
```

```
        fpp1 = 0.5*(0.5*(1/sqrt(2))*D^2 + alphaFlux_X*D);
        fpp2 = 0.5*(0.5*(1/sqrt(2))*E^2 + alphaFlux_X*E);
               %Calculate negative flux terms
        fmm2 = 0.5*(0.5*(1/sqrt(2))*A^2 - alphaFlux_X*A);
        fmm1 = 0.5*(0.5*(1/sqrt(2))*B^2 - alphaFlux_X*B);
        fm = 0.5*(0.5*(1/sqrt(2))*C^2 - alphaFlux_X*C);
        fmp1 = 0.5*(0.5*(1/sqrt(2))*D^2 - alphaFlux_X*D);
        fmp2 = 0.5*(0.5*(1/sqrt(2))*E^2 - alphaFlux_X*E);
               %Calculate positive flux terms at i+1/2 position
        b0 = (fpp1 - fp)^2;
        b1 = (fp - fpm1)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpp = w0*(0.5*fp + 0.5*fpp1) + w1*(-0.5*fpm1 + (3/2)*fp);
               %Calculate positive flux terms at i-1/2 position
        b0 = (fp - fpm1)^2;
        b1 = (fpm1 - fpm2)^2;
        a0 = (2/3)/(epsilon + b0)^2;
        a1 = (1/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fpm = w0*(0.5*fpm1 + 0.5*fp) + w1*(-0.5*fpm2 + (3/2)*fpm1);
               %Calculate negative flux terms at i+1/2 position
        b0 = (fmp2 - fmp1)^2;
        b1 = (fmp1 - fm)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmp = w0*((3/2)*fmp1 - 0.5*fmp2) + w1*(0.5*fm + 0.5*fmp1);
               %Calculate negative flux terms at i-1/2 position
        b0 = (fmp1 - fm)^2;
        b1 = (fm - fmm1)^2;
        a0 = (1/3)/(epsilon + b0)^2;
        a1 = (2/3)/(epsilon + b1)^2;
        w0 = a0/(a0 + a1);
        w1 = a1/(a0 + a1);
        fmm = w0*((3/2)*fm - 0.5*fmp1) + w1*(0.5*fmm1 + 0.5*fm);
               %Combine terms
        fhp = fpp + fmp;
        fhm = fpm + fmm;
        term = X(i) + Y(j);
        term = pi*term;
        term = term/sqrt(2);
            %Calculate L
        L = -((1/DX)*(fhp - fhm)) - ((1/DY)*(ghp - ghm)) + ...
            (sin((X(i) + Y(j))/(sqrt(2))))*(cos((X(i) + Y(j))/(sqrt(2))));
            %Calculate Un+1
        U_new(j,i) = U_2(j,i) + ((2*gamma)/3)*(1/((alphaFlux_X/DX) + (alphaFlux_X/DY) ))*L;
            %Calculate Un+1 with extrapolation factor
        U_new(j,i) = omega*U_new(j,i) + (1-omega)*U_2(j,i);
```

```
            %Increase the number of new Y values
        y_points = y_points + 1;
    end
        %Increase the number of new X values
    x_points = x_points + 1;
        %Reset the number of Y values
       y_points = 0;
end
        %Reset both X and Y values
    x_points = 0;
    y_points = 0;
        %Update iteration count
    iterations = iterations + 1;
        %Calculate L1 norm of successive iterations
    L1norm = max(max(abs(U_new - U)));
        %Update U values
    U = U_new;
end
    %Output iterations
iterations
    %Initialize exact and U variants with only diagonal elements
counter = 0;
for i = 1:numel(X)
    for j = 1:numel(Y)
        if i == j
            counter = counter + 1;
            D(counter) = U(j,i);
            D_ex(counter) = exact(j,i);
        end
    end
end
    %Initialize exact and U variants without shock values
Dm = D;
Dm_ex = D_ex;
    %Shock location
c = 0.1;
    %If the value of X is within 0.1 of the shock
    %then zero out the values of exact and U variants
for k = 1:numel(X)
    if abs(X(k)-c)<=0.1
        Dm(k) = 0;
        Dm_ex(k) = 0;
    end
end
    %Calculate L1 and Linfinity errors
Linf_m = norm(Dm-Dm_ex,inf)
L1_m = (1/numel(U_m))*norm(Dm-Dm_ex,1)
```

Published with MATLAB® R2015a